UNIVERSITY OF OSLO
Department of informatics

Employing Ontologies In Resource-scarce Mobile Ad-hoc Networks - Translating From The Web Ontology Language (OWL) To Topic Maps

Master thesis

60 credits

Ragnhild Juel

2. May 2007

Preface

Working on this thesis has been an interesting journey from beginning to end. The original problem statement was looking into how the ontology language OWL can be used on resource-limited devices for a rescue scenario. I started out by getting familiar with the languages RDF, RDFS, and OWL. Then I started searching for tools for these languages that were lightweight enough to be used on a resource-limited device. It started out optimistically by searching for reasoning engines for OWL. Then, just query engines for OWL, and eventually query engines for RDFS or RDF. There are many tools that are available and that have been created for these languages, but it has been difficult to find what kind of resources they require, and there was no mentioning of them being able to be used on resource-limited devices. I found articles that stated that a reasoner for OWL would not be able to run such a device, but they did not give any detailed information on why. It was only later that we discovered an overview of the complexity of OWL, and with this also an indication of how time demanding reasoning tasks can be. After this, we concluded that having a reasoning engine on a resource-limited device seems impossible, and started looking for other approaches.

The next step was to look at if a translation between OWL and topic maps was possible, since there exists a topic map engine that is designed specifically for resource-limited devices, and if this type of translation still can let us use some of the strengths of OWL like reasoning and restrictions. So I got familiar with the topic maps standard, and looked for translation proposals between OWL and topic maps. The proposals that I found were mainly proposals for a translation between RDF and topic maps, so I started looking at how this could be applied to OWL as well. The translation proposal that seemed to give the best result was by Lars Marius Garshol at Ontopia, and they also had an implementation that was available for download. I got in touch with Garshol and suggested that I could try to look at how his proposal could be used for OWL as well as RDF. He replied that the translation already worked with OWL since OWL is expressed in RDF. Then I started looking for ways in which it was possible to improve the translation between OWL and topic maps. At about this time, I discovered by chance an article describing Pocket KRHyper, which is a reasoning engine for OWL for resource-limited devices, just like I was searching for in the beginning. At this point there was too little time left to explore this reasoner, which would have been very interesting to do. Instead I concentrated on finishing looking at a translation between topic maps and OWL, which has also been very interesting.

I would like to thank my supervisor, Ellen Munthe-Kaas, for guiding me through this thesis, and giving suggestions and help for what step to take next when the preceding step did not lead anywhere. I would also like to thank my family for their love and support, and my friends and boyfriend for their understanding and patience with me when I was trying to explain the difficulties of translation between OWL and topic maps, and they did not have a clue what I was talking about.

Abstract

In a rescue scenario, you have personnel from different organizations cooperating, and this personnel has to communicate both within their own organization and with personnel from other organizations. If they are carrying handheld devices, they can receive and send automatic information updates within a mobile ad-hoc network that consists of all of the handheld devices carried by rescue personnel and sensors that are within range. A challenge with this kind of information sharing is that different organizations may use different data models and vocabularies for defining the same concepts. Ontologies can be used as a bridge between these different vocabularies, enabling a mapping between the concepts of the different vocabularies. The Web Ontology Language (OWL) can be used to express the ontologies.

OWL is a very expressive language, and has the advantage that it is possible to perform reasoning over ontologies and infer knowledge that is not explicitly stated. The only problem is that reasoning engines for OWL typically require a lot of resources, and are therefore not well suited for resource-limited handheld devices.

Topic maps is another technology for expressing ontologies. Topic maps are less expressive than OWL and do not provide support for automated reasoning, but there exists a topic map engine that allows you to browse and query the topic map and that can be used on resource-limited devices.

OWL is built on another language called the Resource Description Framework (RDF), and the topic map and RDF communities have looked at how these two different languages can be translated into each other. Since OWL is built on RDF, we look into if any of the translation proposals for translating between RDF and topic maps also can be used for translating between OWL and topic maps. This will allow us to create an ontology in OWL, perform reasoning on the ontology and add the information inferred from the reasoning to the ontology directly. Then the ontology can be translated into a topic map, and can be used on a handheld device. This thesis looks at how one can translate ontologies from OWL to topic maps, and how usable the resulting topic maps are in a mobile ad-hoc network for a rescue operation.

Index

Preface ... 3
Abstract ... 5
1. Introduction ... 11
2. Problem statement and discussion ... 13
3. Background ... 15
3.1 Definition of an ontology ... 15
3.2 Ad-hoc InfoWare Project ... 15
3.3 The Semantic Web ... 17
3.3.1 Resource description framework (RDF) ... 18
3.3.1.1 Uniform Resource Identifiers (URIs) ... 18
3.3.1.2 The conceptual model ... 18
3.3.2 RDF/XML ... 20
3.3.3 RDF Schema Language (RDFS) ... 22
3.3.4 Web ontology language (OWL) ... 24
3.3.4.1 OWL properties ... 24
3.3.4.2 OWL classes ... 25
3.3.4.3 Reasoning with OWL ... 26
3.4 Topic Maps ... 29
3.4.1 A little history ... 29
3.4.2 The TAO of Topic Maps ... 30
3.4.2.1 Topics ... 31
3.4.2.2 Occurrences ... 32
3.4.2.3 Associations ... 33
3.4.3 Published Subject Indicators (PSI) ... 33
3.4.4 Merging ... 34
3.4.5 XML Topic Maps (XTM) ... 34
4. Related Research ... 37
4.1 Pocket KRHyper ... 37
4.2 SHARK ... 38
5. Examples of OWL ontologies in a rescue scenario ... 39
5.1 Person ontology (pers) ... 39
5.2 Epicrisis ontology (epi) ... 39
5.3 Alertness status ontology (cas) ... 40
5.4 Sensor Upper ontology (sns) ... 41
5.5 Bodysensor ontology (bdsns) ... 41
5.6 Rescue ontology (resc) ... 42
6. RDF/OWL vs. Topic Maps ... 43
6.1 The main differences between RDF and Topic Maps ... 44
6.2 Proposals for translation between RDF and Topic Maps ... 46
6.2.1 Moore's proposal ... 47
6.2.2 The Stanford Proposal ... 47
6.2.3 Ogievetsky's proposal ... 48
6.2.4 Garshol's proposal ... 48
6.2.4.1 Mapping from RDF to Topic Maps ... 48
6.2.4.2 Mapping from Topic Maps to RDF ... 52
6.2.4.3 Implementations ... 53
6.2.5 The Unibo proposal ... 54
6.2.6 Discussion ... 55
7. OWL, Topic Maps and the Ad-hoc InfoWare Project ... 57

8. Using the RTM vocabulary for mapping between OWL and Topic Maps 59
8.1 From RDF to Topic Maps 59
8.2 From RDFS to Topic Maps 60
8.3 From OWL to Topic Maps 61
8.4 Discussion 63
9. Implementation 66
10. Conclusion and further work 70
10.1 Conclusion 70
10.2 Further work 71
11. References 72
Appendix A Ontologies expressed in RDF/XML 77
Appendix B Mappings 86

List of figures

Figure 1: An RDF Graph 19
Figure 2: RDF triple 19
Figure 3: RDF/XML 21
Figure 4: Declaration of class Person using RDFS 22
Figure 5: Creating an instance of class Person 22
Figure 6: Architecture of Pellet 28
Figure 7: Model of the type-instance relationship 32
Figure 8: Model of an occurence 33
Figure 9: A model of an association 33
Figure 10: A type-instance relationship in XTM 1.0 35
Figure 11: How occurrence is expressed in XTM 1.0 35
Figure 12: Expressing an association in XTM 36
Figure 13: The different components of SHARK 38
Figure 14: Person ontology 39
Figure 15: Epicrisis ontology 40
Figure 16: Alertness status ontology 40
Figure 17: Sensor Upper ontology 41
Figure 18: Bodysensor ontology 42
Figure 19: Rescue ontology 42
Figure 20: The standards' families 44
Figure 21: Expressing that an RDF property should be mapped to an occurrence in topic maps. 50
Figure 22: Mappings of RDFS properties to topic maps 50
Figure 23: Mappings of OWL properties to topic maps 52
Figure 24: RDF graph from translation of base name with variant name from a topic map to RDF. 55
Figure 25: Person ontology in topic maps 67

List of tables

Table 1: OWL tractable fragments 14
Table 2: RDF and topic map application areas 43
Table 3: Summary of translation proposals 47
Table 4: Information required and possible mappings to topic map constructs for objects in an RDF statement 49
Table 5: How RDF properties can be mapped to topic maps 60
Table 6: How RDFS properties can be mapped to topic maps 61
Table 7: How OWL properties can be mapped to topic maps 63
Table 8: Statistics for translating from OWL to topic maps 66

1. Introduction

In a rescue scenario, or the scene of an accident, you may have personnel from different organizations that are working together. There may be firemen, police officers, paramedics and physicians. A critical element to the success of a rescue operation is communication, both across and within the different organizations. Today, this communication is oral. If rescue personnel were carrying handheld devices like cell phones or PDAs, these could be used for automatic information updates between them, making communication easier. For instance, if casualties were equipped with sensors monitoring their vital functions, paramedics and physicians could be notified if there was a change in the condition of any of the casualties. Also, information about the rescue scene, like blue prints of buildings or maps, could be distributed to all of the personnel, which would aid them in the rescue operation.

There are a lot of challenges with this kind of communication. The handheld devices are limited in resources with regards to memory, processing power, battery and bandwidth. The communication is done within a mobile ad-hoc network (MANET), which means that different devices may move in and out of range at any time, so you can not rely on having contact with all of the devices at all times. Another issue is in terms of how the information is managed in the network. Since different organizations may use different data models and vocabularies, there is a need for representing the information in a way that everyone can understand.

These are all issues that are considered by the Ad-hoc InfoWare project, a project aimed at creating middleware services for a rescue scenario. This thesis looks at the knowledge management part of the Ad-hoc InfoWare project, or how information is managed. To solve the problems of heterogeneous data models and vocabularies in the different organizations, it is suggested that ontologies are used. To represent the ontologies, the choice is between topic maps and the Web Ontology Language (OWL). The use of topic maps in a rescue scenario has been covered by previous thesis' ([1] and [2]), so for this thesis we have chosen to look into how OWL can be used on resource-limited devices in the context of a rescue scenario. OWL provides a lot more expressivity than topic maps in that you can define restrictions, and it also provides the ability to perform reasoning over the ontology to check for consistency and derive knowledge that is not explicitly stated.

The tools that exist for OWL require a lot of resources, and are not suitable for resource-limited devices. When working on this thesis, a lot of time has been spent looking for tools that can be used with OWL on resource-limited devices. We were not able to find such tools, not until very late, which is covered in chapter 4, so a different approach had to be taken. There does exist a topic map engine that can run on resource-limited devices. OWL is built on another language called the Resource Description Framework (RDF), and the topic map and RDF communities have looked at how these two different languages can be translated into each other. Since OWL is built on RDF, we look into if any of the translation proposals for translating between RDF and topic maps also can be used for translating between OWL and topic maps.

The remainder of this thesis is organized as follows:

Chapter 2 gives a more detailed description of the complexity of OWL that is the cause of why the tools for OWL are so resource demanding.

Chapter 3 gives background information on the Ad-hoc InfoWare Project, RDF and RDFS, the languages that OWL is built on, OWL itself, and topic maps.

Chapter 4 talks about relevant research to this thesis. It describes Pocket KRHyper, a description logic reasoner for resource-limited devices, and the SHARK project that has implemented a topic map engine for resource-limited devices.

Chapter 5 gives an example of ontologies that may be used in a rescue scenario.

Chapter 6 looks at the main differences between RDF and topic maps, and the different proposals that have been made for translating between them.

Chapter 7 looks at how translating ontologies from OWL to topic maps can let us use some of the strengths of OWL in a rescue scenario.

Chapter 8 gives a proposal on how one can translate from OWL to topic maps using one of the proposals described in chapter 6.

Chapter 9 reviews the proposal from chapter 8.

Chapter 10 gives a conclusion and further work that can be done in this research area.

2. Problem statement and discussion

Using the Web Ontology Language (OWL) in an ad-hoc network has been thought of earlier. It is even stated as one of the use cases in the Use Cases and Requirements document [3] for OWL. There it is suggested that OWL can be used for service discovery in an ad-hoc network. Service discovery involves describing and advertising services so that they can be discovered by others. A service is a functionality that a device offers. This functionality can be pretty much anything, ranging from a printer offering printing services or a computer offering internet access, to a thermostat offering temperature readings. OWL can be used to describe the characteristics of different devices, what functionality they offer, what resources they have and policies that they employ. By reasoning over these characteristics, other devices may discover services that they need.

There has been a lot of recent research on how one can use OWL to provide context-awareness in ubiquitous and pervasive computing environments. It has been recognized that ontologies are very well suited for this purpose, since one can use reasoning engines to reason over data and find connections that were not initially known. The only problem with using OWL in such environments is that reasoning engines typically consume a lot of resources, both processing power and memory, and they are therefore not very suitable for resource-limited devices [4]. This is often solved by leaving reasoning to stronger devices like a laptop computer, or a specialized server like they propose in [5].

The reason for why reasoning engines are so heavyweight is the complexity of the models and the reasoning algorithms. [6] gives an overview of the tractable fragments of OWL and their computational properties. They define sets of logics that can be seen as subsets of OWL, and that can handle some interesting reasoning services in polynomial time. The combined complexity, i.e. the complexity with respect to the size of the axioms and the number of facts in an ontology, for OWL DL, OWL Lite and RDF Schema is summed up in Table 1. OWL DL and OWL Lite are sublanguages of OWL, and provide much expressivity, while RDF Schema is much simpler. The different reasoning problems in the table are explained as follows:

- Ontology consistency: Check whether a given ontology has at least one model.
- Concept satisfiability: Given an ontology O and a class A, verify whether there is a model of O in which the interpretation of A is a non-empty set.
- Concept subsumption; Given an ontology O and two classes A, B, verify whether the interpretation of A is a subset of the interpretation of B in every model of O
- Instance checking: Given an ontology, an individual a and a class A, verify whether a is an instance of A in every model of the ontology.
- Conjunctive Query Answering: Given an ontology O and a conjunctive query q, return the answers of the query with respect to O.

We see that the combined complexity for OWL DL and OWL Lite is NEXPTIME-complete and EXPTIME-complete, respectively. To put it very simple, reasoning in OWL DL and OWL Lite can be performed with a worst case time of $2^{p(n)}$ where p(n) is a polynomial function of n, the number of facts in an ontology [7, 8]. For high values of p(n), the worst case time to perform the reasoning may be unacceptable, let alone on a resource-limited device where it may be impossible. In contrast, one can perform concept subsumption and instance

checking in polynomial time using the much simpler language RDF Schema. So there is a big trade-off in expressivity versus computational efficiency.

Language	Reasoning Problems	Combined Complexity
OWL DL	Ontology Consistency, Concept Satisfiability, Concept Subsumption, Instance Checking	NEXPTIME-complete
	Conjunctive Query Answering	decidability still an open question
OWL Lite	Ontology Consistency, Concept Satisfiability	EXPTIME-complete
	Concept Subsumption	EXPTIME-complete
	Conjunctive Query Answering	in 2EXPTIME
	Instance Checking	EXPTIME-complete
RDF Schema	Ontology Consistency, Concept Satisfiability	Trivial
	Concept Subsumption, Instance Checking	In PTIME
	Conjunctive Query Answering	decidable, but complexity bounds not yet established

Table 1: OWL tractable fragments

The Ad-hoc InfoWare project has recognized that the use of ontologies can solve problems of heterogeneity in a mobile ad-hoc network (MANET) for a rescue scenario. In addition, ontologies can be used for profiling and security purposes, assuring that the right people get the right information. Topic maps and OWL have been suggested as languages to describe the ontologies. OWL has the advantage that there exist many ontologies that can be reused, and that one can use a reasoner for consistency checking and inferring new knowledge from an ontology. OWL is also a lot more expressive than topic maps, you can define complex classes and restrictions. Therefore it would be interesting to know how one could use OWL in a MANET for a rescue scenario.

In a MANET for a rescue scenario one can not rely on having a more resourceful machine within range every time there is a need for a reasoner. A MANET implies very mobile nodes which go in and out of range of each other and can create network partitions. One can not rely on infrastructure, like internet connection, to be present at a rescue site since the rescue site may be a remote location or infrastructure may have been destroyed in whatever caused the rescue scenario. Therefore, we need to look for other solutions.

However, there exist tools for employing topic maps on resource-limited devices. This has been covered in master thesis' by Vigdal [2] and Andersen [1] in the context of the Ad-hoc InfoWare project. The topic map and OWL communities have also been interested in finding ways in which one can translate between topic maps and RDF, and have looked at different solutions for if and how this can be done. In this thesis I will look at how OWL ontologies can be translated into topic maps, and if and how this can be used in context of the Ad-hoc InfoWare project.

3. Background

3.1 Definition of an ontology

In this thesis, the word "ontology" will often be used, so I will start out by giving an explanation of what an ontology is.

The word ontology is originally from philosophy where it refers to the subject of existence. In the context of knowledge sharing and artificial intelligence, Tom Gruber [9] defines an ontology as a *specification of a conceptualization.* A conceptualization is an abstract and simplified view of the world and consists of objects, concepts and other entities and the relationships between them that exist in a domain. Simply put, an ontology can be seen as a data model that defines the concepts in a domain of interest and the relationships between those concepts. In [10] Gruber defines the purposes of ontologies as being used for data exchange, unification/translation, calling knowledge services, representing theories, and human communication.

By using common ontologies, agents, or applications, can make ontological commitments. An ontological commitment is an agreement by an agent to use a vocabulary to enable knowledge sharing between other agents. A common ontology that is defined using a formal representation, defines a vocabulary. This vocabulary defines the way in which queries and assertions are exchanged between agents. The agents need not have the same knowledge base, they may have different information about different things, but making an ontological commitment still allows them to share information. An ontological commitment guarantees consistency in how the agents behave, but does not guarantee completeness because agents do not have to provide answers for all possible queries that can be formulated from the vocabulary.

[11] makes a distinction between a domain ontology and an upper ontology. A domain ontology describes concepts in a specific domain or in a specific context. An upper ontology, on the other hand, describes concepts that are applicable across many domains, and are more generalized ontologies. An example of an upper ontology is the Dublin Core Metadata which can be used for describing many kinds of digital material across different domains.

3.2 Ad-hoc InfoWare Project

In this section I will tell a little about the Ad-hoc InfoWare Project. The information presented here is based on articles [12] and [13]. I will not go into details about every aspect of this project, but give a simple overview and describe a bit more thoroughly the part that is relevant to this master thesis.

The Ad-hoc InfoWare Project is aimed at providing middleware solutions for information sharing in a mobile ad-hoc network (MANET) for rescue and emergency scenarios. A MANET is a dynamic and unpredictable network of self-organizing and mobile nodes that can operate without any fixed infrastructure. The nodes are typically heterogeneous devices ranging from cellular phones and PDAs to laptop computers. The devices have different resource constraints such as battery life, processing capacity, memory and bandwidth, all of which need to be considered during run-time. The topology of such a network is not stable because the nodes are mobile, and may move in or out of range at any time. Devices may

even be shut off for a while to save battery. One can therefore not depend on being able to contact any device at any time, and information may be delayed in the network.

In a rescue and emergency scenario there may be an absence of fixed infrastructure because it simply may not be present or it may have been damaged. If the rescue personnel are carrying mobile devices with wireless network interfaces, a MANET could be very useful for aiding them with gathering and sharing valuable information, and also in coordinating the operation and cooperation across organizations. In such a scenario you usually have personnel from different organizations. There may be policemen, firemen, physicians and paramedics that are cooperating. [13] states that there are two central preliminaries for efficient collaboration, which is a critical key for the success of an operation. The first one is the incentive to collaborate, which is naturally given for rescue personnel. The second one is the ability to communicate and efficiently share information, and this is where the middleware services come in.

Rescue personnel need to cooperate and share information both across other organizations and within their own organization. This is a challenge because different organizations usually use different data models and vocabularies to describe the same things. A solution to this is proposed by the use of ontologies.

As mentioned above, different organizations may use different ontologies. To be able to translate between them, for enabling knowledge sharing, shared vocabularies may be used. A shared vocabulary is basically an ontology that describes basic terms in the different ontologies and is used as a bridge between them. The ontologies are proposed expressed using either RDF/OWL or topic maps. They are machine-processable, so a device that receives an information item through the MANET can easily decide whether this piece of information is relevant or not by using the shared vocabulary, and without any user intervention. The shared vocabulary can also be used to query for information throughout the network.

All ontologies are assumed to be created in the a priori phase of a rescue scenario. In a requirements analysis of this project, there has been identified six phases for such a scenario. Phases three, four and five are all performed using resource-limited devices, while in phases one, two and six one can use more resourceful machines. So preparations can be made beforehand to compensate some for the lack of resources in a MANET.

1. A priori
    - This phase takes place before an accident. In cooperation with the authorities, the different organizations exchange information on data formats and any shared vocabularies, and agree upon working methods and procedures.
2. Briefing
    - This phase takes place once an accident has been reported. It involves gathering useful information about the accident site. Some working methods and procedures are agreed upon in accordance with the nature of the accident.
3. Bootstrapping the network
    - This phase takes place at arrival at the accident site. Nodes, or devices, are added to the network, and rescue leaders are appointed.
4. Running of the network
    - This is the main phase of the rescue operation. Information is gathered, shared and distributed, nodes join and leave the network, the roles of the different rescue personnel may change, and ad-hoc inter-organizational groups may form.

5. Closing of the network
   - All network services must be terminated once the previous phase has finished.
6. Post processing
   - This phase involves analyzing the rescue operation, so future operations may be improved.

The Ad-hoc InfoWare middleware consists of five different components that address different issues and challenges in a MANET. In this thesis the focus is on the knowledge management (KM) component. The five different components are listed below.

1. Distributed event notification system (DENS)
   - Handles communication using a publisher-subscriber system.
2. Resource management
   - Keeps track of available resources.
3. Watchdogs
   - Monitors changes and notifies changes made.
4. Security and privacy management
   - Handles secure communication and access control
5. Knowledge management
   - Handles ontologies, metadata and information.

3.3 The Semantic Web

When you search the World Wide Web (WWW) today, the search engines will try to match your search criteria by browsing through HTML (HyperText Markup Language) documents looking for keywords in the text. As a result, you usually end up getting more search results than you are able to process, and many of them are not relevant to what you were interested in. For instance, if you want to find information about the city Paris in France, you enter the keyword “Paris” in a search engine. In the results list, though, you not only find information about Paris in France, but also about many other cities or towns that are named Paris, the heiress Paris Hilton, and the prince of Troy who also had the same name. The search engine is not intentionally trying to confuse you, but the fact is that most of the information on the WWW is intended for human readers, so the computers don’t understand the meaning of it and are not able to give you exactly what you are looking for.

The Semantic Web Activity [14] is an initiative by the World Wide Web Consortium (W3C). One major goal of this initiative is to add semantic meaning to data, so that it is also processable by machines, which will among other things make it easier for the search engines to help you find what you are looking for. Berners-Lee et al. [15] give a good description of what the Semantic Web is:” The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation”.

The Semantic Web Activity wish to accomplish this by creating common standards that can be used across different applications and application domains. By using these standards, automated tools may gather and exchange information easier on the WWW. It may also be possible to extract more knowledge, knowledge that might not be obvious, out from the available information.

The current standards that have been developed by the Semantic Web Activity are the Resource Description Framework (RDF) [16], the RDF Schema Language (RDFS) [17], and the Web Ontology Language (OWL) [18, 19]. These will all be described in the following sections.

3.3.1 Resource description framework (RDF)

RDF is a way to represent information, or metadata, about resources. It was mainly created to represent information about resources on the WWW, but RDF may also be used for other purposes than it was originally intended for. It provides a common framework so that data may be exchanged between applications without loss of meaning.

3.3.1.1 Uniform Resource Identifiers (URIs)

RDF makes use of URIs, or actually URI References, to define resources. A URI Reference is a URI with an optional fragment identifier at the end, separated by a "#". A common URI is a Uniform Resource Locator (URL), which is a subset of URIs. A URL is used to identify a resource that exists on a network, and is used by web browsers to retrieve this resource. But where a URL only defines resources that are available over a network, a URI may define many other things as well. It can be used to define physical things like living creatures, businesses, food, or more abstract concepts that you can't touch or feel. RDF does not expect to be able to retrieve resources, or even just the definition of these resources, over the internet based on their URI Reference. Neither does it assume any relationships between resources if their URI Reference is similar. In RDF, the URI Reference is only used to identify a specific resource, so that if two resources have the same URI Reference, RDF can assume that the two resources are actually the same.
A set of URI References that are intended for a specific purpose, is called a *vocabulary*.

3.3.1.2 The conceptual model

With RDF you can describe a resource in a similar way that you would do in natural language. RDF is based on the use of statements that consist of a *subject*, a *predicate*, and an *object*. The subject is the resource that is being described, the predicate is the property that describes something about the resource, and the object is the value of the property. The subject and the predicate both need to be defined by a URI Reference. The object can either be the URI Reference of another resource, or a *literal*. For instance, if you want to describe a parent-child relationship between two people, both the subject (the parent) and the object (the child) would have to be identified using a URI Reference. But for some properties, like the name of a person, the value, or object, is just a string of characters, so it will be represented as a literal.

There are two types of literals that are used in RDF, the *plain literal*, which is basically a string of characters, and the *typed literal.* When you use the typed literal, you also need to specify a URI Reference for the datatype that the literal is supposed to be interpreted as. RDF does not have any built-in datatypes of its own. For instance, if you want to specify the date that a person was born, you could just use the plain literal "`1968-05-15`". But then it would be difficult for applications to recognize that this information should be interpreted as a date. So instead you can use a typed literal like this one:
"`1968-05-15`"^^ `http://www.w3.org/2001/XMLSchema#date`, where `http://www.w3.org/2001/XMLSchema#date` is the URI Reference that contains the

definition of the datatype that we want to relate to the string "1968-05-15". http://www.w3.org/2001/XMLSchema# is a vocabulary that also contains the definitions of many other datatypes, and is commonly used for this purpose. By using common URI References for datatype definitions, datatypes may be interpreted equally across applications.

URI References can be very long and eventually cumbersome if you repeat writing them many times. To solve this, a shorthand notation may be used by using an XML *qualified name*. A qualified name consists of a prefix that has been assigned to a namespace and a local name separated by a colon. For the vocabulary containing different datatypes that was mentioned above, we could assign the prefix xsd to represent the namespace: xmlns:xsd="http://www.w3.org/2001/XMLSchema#". Now when we need to refer to the date datatype, we don't need to write the entire URI Reference, but only xsd:date. Below is another prefix assigned to a namespace for a vocabulary that will be used in the following example.
xmlns:pers="http://www.ifi.uio.no/dmms/epicrisis_ex/person#"

Imagine that you are in a position where you want to describe a patient at a hospital. To describe this patient you might use properties like name, social security number, address, gender, and many other things. All of these properties that are needed to describe a person have been defined in the vocabulary pers. A particular patient has been defined as http://www.ifi.uio.no/dmms/epicrisis_ex/individualEx#P25, and this person has social security number, or property pers:hasPersonId, "23098234125". To represent this information in RDF, you can use an RDF *graph,* which is the conceptual model of RDF, or a *triple,* which is a shorthand notation. Figure 1 shows how a graph could be used to represent the information about the patient.

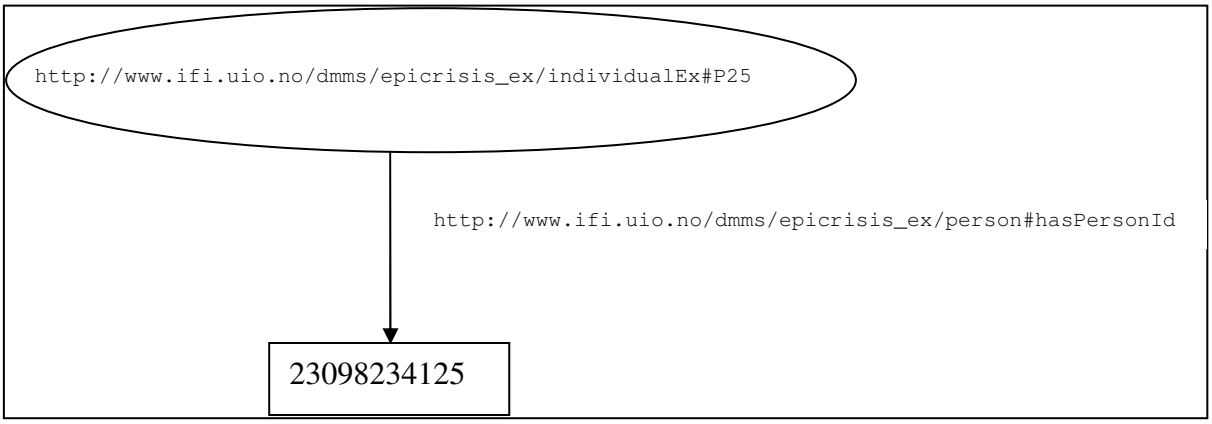

Figure 1: An RDF Graph

An RDF graph consists of a node for the subject and a node for the object, and the predicate is a directed arc from the subject to the object. In the graph, all the URI References have to be spelled out completely. Furthermore, a distinction is made in the graph between the nodes that are URI References, which are drawn as ellipses, and those that are literals, which are drawn as boxes. Figure 2 displays the triple notation describing the same information as in the graph:

```
http://www.ifi.uio.no/dmms/epicrisis_ex/individualEx#P25 pers:hasPersonId "23098234125"
```

Figure 2: RDF triple

As you can see, the triple notation only consists of the subject, predicate and object. In this notation you can use the XML qualified name, you don't have to spell out the entire URI Reference.

A statement in RDF can only link two resources together, the subject and the object. Sometimes one wants to make a statement linking a resource to a group of resources. For instance, if one wants to state that a physician has a group of patients or that a book has several authors. This can be done by creating multiple statements with the same subject and predicate, but different objects. A much simpler way is to use the container types that are built into RDF. RDF has defined three different container types: `rdf:Alt`, `rdf:Seq`, and `rdf:Bag`. The prefix `rdf` represents the URI "`http://www.w3.org/1999/02/22-rdf-syntax-ns#`" which is the vocabulary that defines RDF itself. The `rdf:Bag` is used to represent a group of resources or literals where the sequence order of the members of the group does not matter. The `rdf:Seq` is like `rdf:Bag` except for that in this case the sequence order of the group members is important. `rdf:Alt` is used for representing a group of resources where the members of the groups are all alternatives to something, usually the value of a property.

When using any of the container types mentioned above, one can not for certain say that the members of a container are the only possible members of that container because there may exist other vocabularies that add additional members to it without ones knowledge. This is because there is no way to close the containers, i.e. say that the members that are listed are the only allowed members and that it is not possible to add more members. If it is desirable to create a closed group, one can instead use a collection in RDF. An RDF collection is basically a linked list of resources where the last resource is always the predefined `rdf:nil` which denotes that it is the end of the collection, and it is not possible to add any more resources to the list.

RDF does not restrict the number of statements that include the same subject and predicate. It is normal for people to only have one social security number, at least within the same country, but this restriction is something that cannot be stated in RDF. Furthermore, RDF does not understand the actual meaning, or what the creator of the vocabulary intended as the meaning, of a predicate used in a statement. But if common vocabularies are used, applications can be carefully programmed to look like they understand the meaning.

3.3.2 RDF/XML

The RDF graph and triples provide a way of modelling RDF in a way that humans easily can understand, but they are not very useful for exchanging data between different applications. For this purpose RDF/XML can be used. RDF/XML is an XML notation for exchanging the same information that is provided in an RDF graph. Some may think that RDF/XML adds too much overhead, and that one can just use XML instead. One of the main differences between regular XML and RDF/XML is that RDF/XML allows you to combine information from several different sources. In regular XML, information about a resource has to be bundled together for applications to understand that the information is about the same resource. In RDF/XML URI References are used to identify the resources. So it is possible to have different information about the resource scattered over different files, and applications will still be able to interpret that the information is about the same resource.

The graph in Figure 1 can be written in RDF/XML as seen in Figure 3.

```
<?xml version=”1.0”?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:pers="http://www.ifi.uio.no/dmms/epicrisis_ex/person#">

<rdf:Description
    rdf:about="http://www.ifi.uio.no/dmms/epicrisis_ex/individualEx#P25">
  <pers:hasPersonId>23098234125</pers:hasPersonId>
</rdf:Description>
</rdf:RDF>
```

Figure 3: RDF/XML

First, at the top, is the XML declaration which is necessary to inform that the following will be written in XML, and what version of XML is used. After that, is the start tag for RDF, `<rdf:RDF>`. This tag states that the following information should be interpreted as RDF. Within this tag are the namespace declarations that you need. The first namespace declaration is the RDF vocabulary, and the declaration states that all tags that begin with `rdf:`, are parts of this vocabulary. The second namespace declaration is for the previously mentioned vocabulary containing properties used for describing a person, and all tags that begin with `pers:` should be taken from this vocabulary. After the `<rdf:RDF>` tag, with its namespace declarations included, is where the fun starts, and where you can start to describe resources. The `<rdf:Description>` tag starts a new description of a resource. The URI Reference of the subject is defined in the `rdf:about` parameter of the `rdf:Description`. After this starting tag, you can create as many statements about the resource that you want to. If you want to make multiple statements about a resource, you may either do it within the same `<rdf:Description></rdf:Description>` tags, or create a new set of `<rdf:Description> </rdf:Description>` tags for each new statement. The object of the statement is surrounded by a start tag and an end tag for the predicate in the statement. If the object is a URI Reference instead of a literal, the `rdf:resource` element is used to define the URI. The end tag `</rdf:Description>` marks the ending of this specific resource description. Finally, the end tag `</rdf:RDF>` states that this is the end of the information that should be interpreted as RDF.

3.3.3 RDF Schema Language (RDFS)

So far in the examples for RDF I have only assumed that the vocabularies for the description of classes and properties exist, and the example only uses these vocabularies to describe a resource. These vocabularies have to be made, of course, and the RDF Schema (RDFS) language is what is used to create them. I will not go into details on how to use RDFS, but just give a small introduction with an example.

I will continue to use the example from the previous section, and define the class `Person` by using the `rdfs:Class` element, and the property `hasPersonId` by using the element `rdf:Property`. See the example in Figure 4. This is very similar to the RDF/XML that was described in earlier, just with the extra namespace declaration for `rdfs` so that we can use the elements from that vocabulary. The `xml:base` declaration identifies the current document.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.ifi.uio.no/dmms/epicrisis_ex/person">

<rdfs:Class rdf:ID="Person" />

<rdf:Property rdf:ID="hasPersonId" />

</rdf:RDF>
```

Figure 4: Declaration of class Person using RDFS

It is also possible to assign a *range* and a *domain* for a property using `rdfs:Range` and `rdfs:Domain`. The domain of a property is the specification of what class has to be the subject, and the range is the specification of the object. One has to be very careful about setting the range and the domain of a property. If the domain of a property is set to be a class called Person, it states that only instances of the class Person may use this property. But if an instance of a class called Dog uses this property, then it is assumed that this instance must also be an instance of class Person, which would not be correct.

By including the vocabulary created above, you can create an instance of the class `Person`. The example in Figure 5 creates an instance, called "`P25`", of class `Person`.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:pers="http://www.ifi.uio.no/dmms/epicrisis_ex/person#"
  xml:base="http://www.ifi.uio.no/dmms/epicrisis_ex/individualEx">

<pers:Person rdf:ID="P25" />

</rdf:RDF>
```

Figure 5: Creating an instance of class Person

By assigning an id to the instance using `rdf:ID`, external resources may reference this instance by creating a URIref by combining the base URI and the ID. It is also possible to create an anonymous instance, in other words, an instance that cannot directly be referenced externally. By using the `rdf:nodeid` instead of `rdf:ID`, you state that the ID is only relevant internally, and cannot be directly used by external resources.

3.3.4 Web ontology language (OWL)

OWL adds more expressivity to RDF and RDFS, and the interchange syntax of OWL is also RDF/XML. The namespace for OWL is "http://www.w3.org/2002/07/owl#", and the prefix `owl` is commonly assigned to this namespace. Using OWL you can define classes and properties more detailed in form of the different relationships you may have between different classes, and use restrictions on properties. With OWL you can join different ontologies together just like RDF can join different vocabularies together. This makes it a powerful tool because you can reuse and combine many already existing ontologies, so you don't always have to create your own. When creating an ontology, you can add metadata to the ontology itself by expressing what version an ontology is, using `owl:versionInfo`, or link to a previous version using `owl:previousVersion`. You can also state whether the ontology is compatible or incompatible with other versions using `owl:backwardCompatibleWith` and `owl:incompatibleWith`. Some applications, called *reasoners,* are able to derive information that is not explicitly stated in the ontology by following the semantics of OWL.

OWL has three sublanguages: *OWL Lite, OWL DL* and *OWL Full*. The difference between them is the degree of their expressiveness. Choosing what sublanguage to use depends greatly on what you want to use it for. The sublanguages are subsets of each other. This means that every legal OWL Lite ontology is a legal OWL DL ontology, and every legal OWL DL ontology is a legal OWL Full ontology.

OWL Lite is the simplest of the sublanguages, providing a low degree of expressiveness, and can be used for a simple classification hierarchy. The basic features of OWL Lite is that it can be used to express simple class hierarchies, you can create restrictions, but cardinality restriction may only be 1 or 0, and you can express a class as the intersection of named classes or restrictions.

OWL DL is the middle way between OWL Lite and OWL Full. It provides you with a lot of expressivity, and you are guaranteed that a computation will finish in finite time. DL stands for *description logics* which is a field of research that deals with a particular decidable fragment of first order logic. OWL DL is given its name because of its close relationship with description logics, and because of this relationship it has desirable computational properties for reasoning systems. OWL DL allows the use of more set operators than OWL Lite, you can express that a class is the union of or complement of other classes. It also allows more complex statements of class descriptions, and the cardinality restrictions can use any non-negative number.

OWL Full is the most expressive language of the three. The main difference from OWL DL to OWL Full is that classes and properties are also allowed to be individuals. Although you can create complex data structures and express almost anything you want to, you are not guaranteed that a computation by a reasoner will finish in finite time.

In this thesis I will concentrate mostly on OWL DL because it provides the most expressiveness with a guarantee that computation will finish in finite time.

3.3.4.1 OWL properties

Properties in OWL, like in RDF, are binary relations that link two individuals together. There are two types of properties, *datatype properties* and *object properties*. A datatype property links an individual to a literal, and an object property links two individuals together. These

properties are similar to relations in UML and similar modelling languages. A property can be a subproperty of another, but then both properties have to be either datatype or object properties. For instance, a property called `hasParent` could have `hasMother` and `hasFather` as subproperties. You can also create inverses of properties, meaning that if property `p1` defines a relationship between individuals `a` and `b`, then the inverse property would define a relationship between `b` and `a`. This is similar to having two-way links in a UML diagram.

You can define the range and the domain of a property, just like you can in RDF, but this needs to be done carefully, as I have mentioned before.

Properties can be defined to have certain characteristics, or be of a certain type. They can be *functional*, *symmetric, transitive,* and *inverse functional.* If individual `A` uses a property that is functional, it means that individual `A` can link to at most one other individual using this property. For a property that is inverse functional, it is the other way around. An individual can only be linked to by another individual once using this property. Symmetric properties are their own inverses. This means that if the property links individual `A` to individual `B`, then it also links individual `B` to individual `A`. A transitive property means that if a property links individual `A` to individual `B`, and individual `B` to individual `C`, then we can infer that the property also links individual `A` to individual `C`.

3.3.4.2 OWL classes

The `owl:Class` construct that is used to define a new class in OWL, is defined as a subclass of `rdfs:Class`, but in OWL only `owl:Class` is used. A class in OWL can be seen as a description of a set containing individuals. A class in OWL can have zero or more subclasses, and one or more superclasses. Every class is automatically a subclass of the predefined `owl:Thing`, which is the class of everything.

A class can have multiple superclasses, and the individuals that can belong to a class is the intersection of the individuals that belong to that class' superclasses. In addition to this, a class can be defined using set operators on other classes. You can define a class as being the union of other classes by using the `owl:unionOf` construct, or the intersection of other classes by using `owl:intersectionOf`. When defining that a class is a union of other classes, you are stating that the class contains the union of all individuals in the other classes. The `owl:complementOf` construct can be used to define a class that contains all the individuals that do not belong to another class. By using these set operators, one can create rather complex classes.

Properties can be used to create restrictions for the individuals that belong to a class. The restrictions are usually made up of a property and a condition for this property. They describe an anonymous class that consists of all the individuals that satisfy the restriction. A class is defined as a subclass of its restrictions, so if you define a set of restrictions for a class, it is the intersection of the individuals of the anonymous restriction classes that can be allowed to be an individual of that class. There are three main categories for restrictions: *quantifier restrictions, cardinality restrictions,* and *has value restrictions.*

A quantifier restriction consists of a *quantifier*, a property and a *filler*. A filler is a class or literal that you want to restrict the property to. For instance, if you wanted to specify that the

`hasPersonId` property only could have integer values, the filler would be the URI Reference to the integer datatype definition. The quantifier can be either *existential* or *universal*. An existential quantifier can be read as "some of" or "there exists", and uses the `owl:someValuesFrom` property. If this quantifier is used in a restriction for a class, then an individual of that class must have a relationship along the given property to an individual of the given filler. A universal quantifier uses the `owl:allValuesFrom` property, and means that if a class has this restriction, then for the individuals of that class, all the relationships along the property in the restriction must be to an individual that is specified by the filler. A universal quantifier does not specify that a relationship exists, though, only that if it does exist, then it must be to an individual that is a member of the filler. If you want to specify the existence of a relationship as well, you would have to create an additional restriction with the existential quantifier.

A cardinality restriction states how many relationships an individual of a class may have along a specific property. The restriction may specify a minimum, maximum or exact value. For instance, if you wanted to specify that a person can only have one social security number, you would create a restriction for the class `Person` for the property `hasPersonId` to have cardinality exactly 1 using the `owl:cardinality` property. When specifying the minimum or maximum cardinality you would use the properties `owl:minCardinality` or `owl:maxCardinality`.

The "has value" restriction consists of a property and another individual or datatype value. It specifies that an individual of a class with this restriction has to have at least one relationship along the given property to the specified individual.

Restrictions may be *necessary* or *necessary and sufficient* [20]. A necessary restriction states that an individual may only be a member of this class if the restriction is satisfied. A necessary and sufficient restriction states the same as a necessary restriction, but additionally it means that any individual that satisfies the restriction must be a member of the class. In other words, it qualifies an individual to be a member of the class. If a restriction on a class is necessary, the class is expressed as being the subclass of the restriction. If a restriction is necessary and sufficient, the class is expressed as being an equivalent class to the restriction.

3.3.4.3 Reasoning with OWL

I have previously mentioned that one can use reasoners with OWL. Reasoners are also called reasoning engines and inference engines. The standard tasks you can perform with a reasoner are as following ([21]):

- **Concept satisfiability testing**: Checks if the descriptions of a class in an ontology are contradictory and causes the class to not be able to have any instances. For instance, if a class is a subclass of two other classes that are disjoint, it is impossible for the class to have any members. This is because an instance of a subclass is automatically an instance of the superclass, and two classes that are disjoint are not allowed to have any instances in common. A class that can not have any instances is considered to be inconsistent.
- **Concept subsumption testing**: Finds information in the ontology that is not explicitly stated. This is done by following the descriptions of properties and classes. It can be used to infer equivalences between classes, or infer a more generalized/specialized class hierarchy.

- **Consistency checking/Knowledge base satisfiability testing**: Tests whether the ontology contains any contradicting facts. Creating an instance of a class that can not have any instances is an example of a contradictory fact.
- **Instance checking**: Infer that instances belong to certain classes without this being explicitly stated in the ontology.

In addition to these standard tasks, reasoners often provide query support.

Since OWL DL and description logics are very closely related, most reasoners for OWL are actually description logic reasoners. Most description logic reasoners can handle computation for OWL DL, and therefore also for OWL Lite, but it is not a matter of course that they can handle OWL Full because it goes beyond the bounds of description logics. The description logic reasoners that can be used with OWL either have support for OWL directly or they often have something called a DIG interface [22]. Reasoners have usually been very language dependent, with Lisp being the most common programming language. To make the reasoners available to applications written in other, more popular, languages as well, the DIG interface was created. The DIG specification is really just an XML schema for a description logic concept language with ask/tell functionality. It allows ontology editors to export OWL ontologies to the format defined by the XML schema, and communicate with an external description logics reasoner using the HTTP protocol. Using ontology editors that support DIG, which most do, one can choose whichever reasoner one wants to just as long at it also supports DIG. It is a "plug and play" for description logic reasoners.

Some examples of popular reasoners that can be used with OWL and that all support DIG are Pellet [23], RacerPro [24], FaCT++ [25], and KAON2 [26].

Pellet is an open source reasoner implemented in Java. It was the first reasoner to provide full support for OWL DL. It supports OWL directly because it designed for that purpose from the beginning, and it also supports the DIG interface. Since it was designed for OWL, it also provides interfaces for some popular RDF/OWL toolkits like Jena [27], which provides a Java programming API for manipulating RDF and OWL. Figure 6 shows the architecture of the main components of Pellet. As you can see in the figure, Pellet supports the query language SPARQL[1].

[1] A query language for RDF graphs created by W3C.

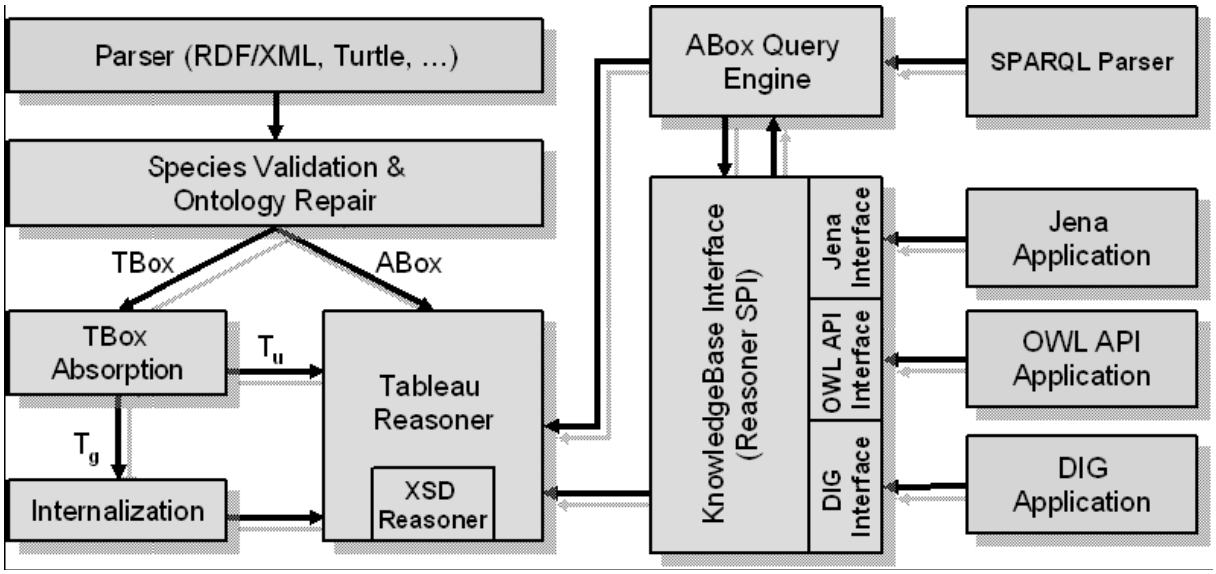

Figure 6: Architecture of Pellet

RacerPro is a commercial reasoner, but it is possible to download a trial version for a limited period of time. It supports OWL DL ontologies, and provides access from Java and Lisp using IP sockets. RacerPro supports queries that use the query languages OWL-QL[2] or SPARQL.

FaCT++ is an open source reasoner for OWL DL implemented in the programming language C++.

KAON2 is an infrastructure implemented in Java that is used for handling ontologies that are expressed in OWL DL, F-Logic and SWRL. It provides programming APIs for all of these languages, and supports queries written in SPARQL. KAON2 is free of charge for universities and noncommecial academic usage, but for other usages there exists a commercial version called OntoBroker OWL.

[2] OWL query language. Implemented by Stanford Knowledge Systems Laboratory.

3.4 Topic Maps

Topic maps is a standard that can be used for creating ontologies. The topic maps standard was created to solve the problems of information retrieval on the WWW. It is based on ideas from traditional indexing, library science and knowledge representation combined with modern linking and addressing techniques. [28] states that the purpose of the topic map is to "convey knowledge about resources through a superimposed layer, or map, of the resources. A topic map captures the subjects of which resources speak, and the relationships between resources, in a way that is implementation-independent."

A popular quote, whose source is unknown, within the world of topic maps is "a book without an index is like a country without a map". When travelling to a destination that you are not yet familiar with, you can either travel around for a while and hope that you eventually get there, or you can use a map that tells you how to get there. There is little doubt that most people would prefer to use a map. It will be even easier to find your way if you are using a Global Positioning System (GPS) that tells you where you are at all times, so the chances of getting lost are minimal. Similarly, if you are looking for information in a book, you may either read it from cover to cover to find what you are looking for, if the information is even present in the book. Or, you can look in the index to see if the subjects that you are looking for are covered, and if they are, where you can find them. One has to admit that the latter approach will save you a lot of time. An index in a book can be seen as an information map of that book.

In a book, the index covers all the information that can be found in the book. On the WWW, however, traditional indexing is not so simple. One can of course make simple indexes on the individual documents that are present on the WWW, but that does not help when information spans across many documents. That would require a merging of indexes that could span over very large amounts of information. If computers were to mechanically create a regular text index on the WWW, it would include almost every single word of every document, which would leave an index that is not very feasible to use. It would also have problems with synonym and homonym words, where different words have the same meaning and the same word has different meanings, respectively. In other words, there would be a lack of intelligence as to what the documents are actually about. Unfortunately, this is close to how most search engines for the WWW are built today.

Topic maps aim to be the equivalent of a back-of-book index for electronic documents. Pepper [29] writes about topic maps that "It is our firm conviction that they will become as indispensable for tomorrow's information providers as maps for the traveller. And once topic maps have become ubiquitous, they will indeed constitute the GPS of the information universe".

3.4.1 A little history

Guide to the topic map standards [30] gives a good overview of the process of standardization of topic maps. The International Organization for Standardization (ISO) [31] was the organization that started working with the standardization of topic maps. More specifically, it was the subcommittee SC 34 within the organization. SC 34 works with among other things SGML, HyTime and topic maps.

The syntax for topic maps became a standard known as ISO 13250:2000 [32]. The syntax used is known as HyTM (HyTime Topic Maps) because it uses the standard ISO 10744 HyTime for linking and addressing. Wikipedia [33] defines HyTime (Hypermedia/Time-based Structuring Language) as a markup language that is an application of SGML [34] (Standard Generalized Markup Language). SGML is a standard for defining markup languages. I will not go into details about HyTime and SGML because they are both rather complicated standards, but it is good to know that the well-known HTML originally was designed based on SGML, and later evolved into an SGML application. XML is a subset of SGML, designed to be a simpler standard for general-purpose usages. HTML and XML also borrow concepts from HyTime, so the standards are closely linked together.

There were some issues that were recognized as problematic with the HyTM syntax used. One of these issues was that most people were using XML as a markup standard, not SGML, and therefore an XML version of the topic map standard was needed. The result is a standard called XTM, an abbreviation for XML Topic Maps, and is the version that is most used today for interchanging topic maps. XTM will be covered in more detail later.

The topic maps standard will be revised into a new multi-part standard. The new standard will be divided into these five parts:

1. *Topic Maps - Overview and Basic Concepts* [35]. This document is currently a draft, and it is not finished. It is supposed to give a simple introduction to topic maps and its concepts for people who are interested to learn about topic maps without all the technical specifications.
2. *Topic Maps - Data Model* [36] (TMDM), formerly known as the *The Standard Application Model* (SAM). The TMDM defines a formal data model for topic maps and is the basis on which XTM will be built on.
3. *Topic Maps – XML syntax* [37]. This is the definition for the interchange syntax for topic maps based on XML. It also provides a mapping between the data model and the XML syntax.
4. *Topic Maps – Canonicalization* [38]. This part of the standard defines an algorithm for canonicalization of an instance of the TMDM.
5. *Topic Maps Reference Model* [39]. This part describes a more abstract model, and talks about subject maps instead of topic maps.

In addition to this, it has been decided to create two more standards, a topic map query language (TMCL) and a topic map constraint language (TMCL). These two standards will both be based on the TMDM.

3.4.2 The TAO of Topic Maps

This section is based on information found in [29], [36] and [28]. There is no specified graphic notation for topic maps, so the models used in this section are generic models that do not follow any special notation rules, but are only used for clarification of the different concepts. It will be explained how to interpret the different models.

TAO is an abbreviation for topics, associations and occurrences, the concepts that are present in a topic map. The concepts are taken from the world of indexing. In a regular back-of-book index, you have an alphabetical list of topics with references to the occurrences of those topics in the book. In some cases you also have associations between different topics that are denoted by the words “see also”. The different concepts will be explained following.

3.4.2.1 Topics

A topic is a representation of a subject in the real world. [32] defines a subject like this: "a 'subject' is any thing whatsoever, regardless of whether it exists or has any other specific characteristics, about which anything whatsoever may be asserted by any means whatsoever". So a subject can basically be anything you want it to be, or anything that you would like to represent in a topic map. A topic is said to "reify" a subject. To reify is to treat an abstract concept as a material thing, so you can make assertions about it as if it actually was the real thing. If a topic map is consistent, there is a one-to-one relationship between subject and topic. In other words, a subject can only be reified by one topic. In a topic map document, however, you can have several topics reifying the same subject, but this should be done in a way that the topics can later be merged to form a consistent topic map.

The idea of identity is very important in the topic map paradigm because the goal is to have a one-to-one relationship between subject and topics. The reason for this goal is so all knowledge about a subject can be accessed through a single topic. It is therefore important to have an unambiguous identity of a topic. In a topic map, you use URIs to identify the different topics. At least one identity must be used, but it is also possible to use several. A topic can have two different types of identity: a *subject locator* or a *subject identifier*. In a topic map, you can have topics that reify subjects that are *addressable*, i.e. they can be directly addressed within the world of computers. An example of an addressable subject is a web page. In this case, a subject locator is used, and the URI is the address of the subject. In many cases you also have subjects that are *non-addressable*, for instance a human being, a concept, or the moon. These are still valid subjects in a topic map, and must be given some sort of identification. In this case, *subject indicators* and subject identifiers are used. A subject indicator is a resource that is supposed to unambiguously define a subject. A subject identifier is the locator, or the address, of the subject indicator. If you have a topic that represents a specific person, the subject indicator may be a web page that describes the person, and the subject identifier would then be the URL of that web page.

When using subject indicators, you may have a situation where topics that are meant to represent the same subject, have used different subject indicators. These two topics will not be merged even though they represent the same subject. A solution for this may be using a third topic that establishes identity through both the subject indicators. In this way, topic maps can be used for mapping between different ontologies.

To define what kind of a topic you have, you can use a *topic type.* A topic type is basically just another topic that defines the type of a topic in a *type-instance* relationship. We can use the example from 3.3.3, where we want to specify that the individual P25 is a person. To do this, we would have to create two topics, one representing the individual P25 and one representing the concept of a person. Then we would have to create a type-instance relationship between them. Figure 7 shows a model displaying the two topics represented by ellipses, and a relationship between them stating that P25 is the instance and Person is the type. A type-instance relationship is not transitive. This means that if a topic "B" is of type "A", and topic "C" is of type "B", it does not follow that topic "C" is also of type "A". A topic may have zero or more topic types.

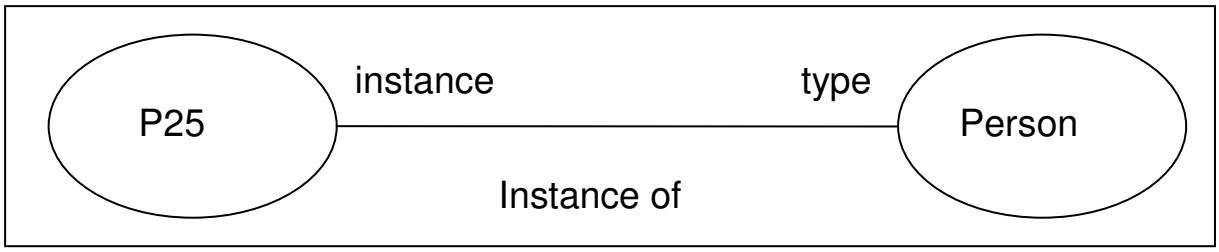

Figure 7: Model of the type-instance relationship

Topics can have three types of characteristics: names, occurrences and roles played in associations. All of these characteristics are only valid within a certain scope. Scope is used to assign a context to the characteristics of a topic. What kind of context is not specified in the standard, but is up to the author of the topic map. For instance, you can have different scopes for different languages. This way, you can have an application that supports internationalization by choosing a different scope based on the user's language preference. Scope can also be used if you have different contradictory statements about a topic by assigning the appropriate scope to the different statements. It is then up to the user or the application to decide which statement it more valid based on the scope. If a scope is not given explicitly for a characteristic, the characteristic is in the *unconstrained scope.* That means that it is always valid, no matter what scope is used. Scopes are defined by *themes*, which are just topics that specify the scope.

A topic may have zero or more names. A name consists of a base name, which is always a string, and zero or more variant names. Variant names may be a string or any other type of resource. A variant name may have zero or more parameters, specified by a set of topics, which can be used by an application to decide what name to use. For instance, a variant name can be an abbreviation of the base name, and the application can then decide whether or not is more appropriate to use the full name (the base name), or the abbreviation. The topic naming constraint states that two different topics can not have the same name in the same scope. If they do, they are to be considered equal and should be merged.

3.4.2.2 Occurrences

Occurrences are another characteristic of topics, and are therefore valid only within a certain scope. An occurrence is an information resource that is linked to and relevant to the topic in some way. A topic name is a specialized sort of occurrence. The information resource is either addressable by a URI, which makes it a "resource reference", or it can be inline data, "resource data". A resource reference type of occurrence can be for instance a link to an article or a photo that is relevant to the topic. Resource data is a string that represents some information about the topic. It can be a date, a short description of the topic etc. Just like topics can have types, so can occurrences.

Using the example from section 3.3.1.2, we want to state that individual P25 has social security number "`23098234125`". To do this, we would create a topic representing individual P25, an occurrence that has "`23098234125`" as its value, and another topic representing the occurrence type "hasPersonId". Figure 8 shows a model representation of this example, where the ellipses are topics, and the square is an occurrence.

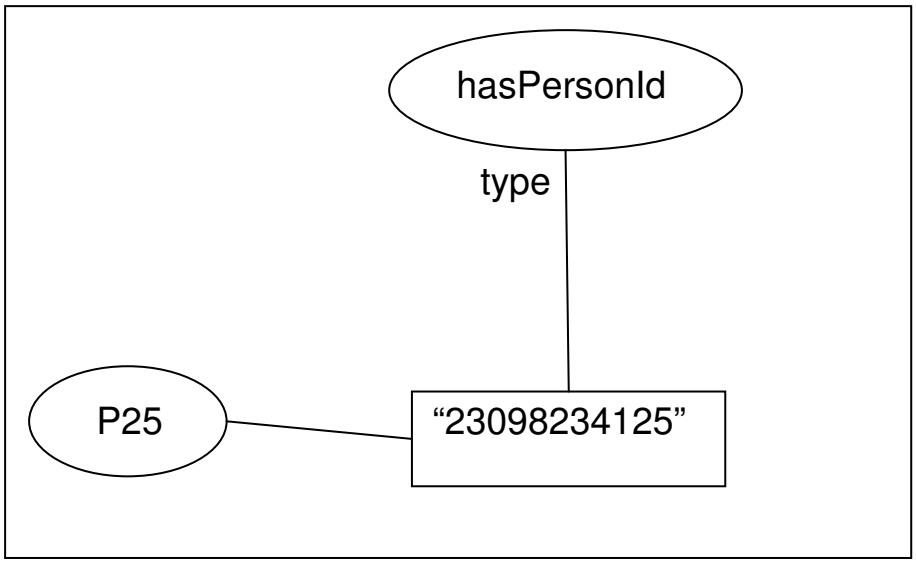

Figure 8: Model of an occurence

3.4.2.3 Associations

Associations are relationships between two or more topics. The topics that are a part of the relationship expressed by the association are *members* of that association. There is no directionality in an association; instead, *association roles* are used to express how the members are involved in the association. Association roles are also expressed as topics. Associations can also have types defining what kind of association it is. A specialized type of association is the *type-instance* relationship.

For instance, if we wanted to say that our person P25 was the husband of another person, P26, we could create a topic for each of them, create an association of type “married” between them, where P25 has the role of husband, and P26 has the role of wife. The association roles also need to be represented by topics. Figure 9 shows a model of the example.

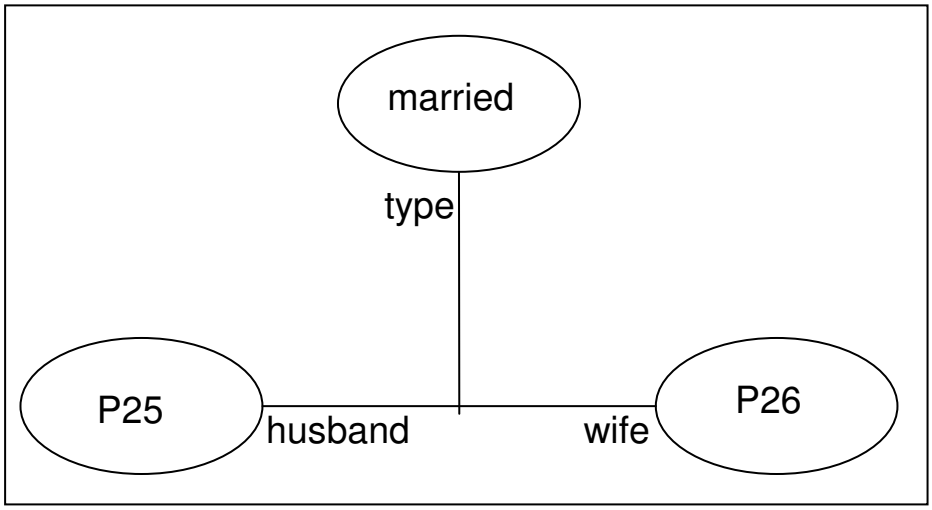

Figure 9: A model of an association

3.4.3 Published Subject Indicators (PSI)

To best identify a topic that is non-addressable, in order to improve the interchangeabililty and mergeability of topic maps, it is recommended to use PSIs if possible. PSIs are subject

indicators that represent specific subjects and are maintained at an advertised address for easy access for everyone. PSIs are used to prevent non-addressable subjects to have many different subject indicators, perhaps because they all have different authors, which will make it difficult to find out if two topics actually represent the same subject. An example of a PSI is the association type and the corresponding association role types that represent the subtype-supertype relationship.

3.4.4 Merging

Previously I have mentioned that topics that reify the same subject such be merged in order to create a consistent topic map. It is assumed that two topics reify the same subject if they have one or more subject indicators in common, or reify the same addressable subject, or have the same base name in the same scope. The result of merging two topics is a single topic whose characteristics are the union of the two original topics' characteristics. Merging two topic maps is the same as merging all the topics in the two topic maps that represent the same subjects.

3.4.5 XML Topic Maps (XTM)

There are currently two versions of XTM, XTM 1.0 [28] and XTM 2.0. The latter is out for review to become an ISO standard, and is considered to be stable as no more changes are expected to be made. [37] has a complete list of the differences between the two versions. Many of the differences are just changes in names of element for simplicity sake. [40] explains the reasons for the changes that were made. When creating the first version, they had ten specific design goals. When creating the second version, they only had one goal, to be able to transfer topic maps from one place to another. So I guess one could say that XTM 2.0 is trying to be a simpler specification than XTM 1.0. I will concentrate on XTM 1.0, since it is the version that is currently most used.

I will not go through every element type of XTM 1.0, but show how the examples from the previous sections can be expressed, just to give an indication of what it looks like. For simplicity, I will use the same URIs as in section 3.3.1 where it is applicable.

Figure 10 shows how we can express that individual P25 is of type Person. First, we create a topic representing the concept of a person within the first `<topic>` element. The `id` attribute of the `<topic>` element gives the unique id for this topic. The `<subjectIdentity>` element specifies what subject is represented by this topic. To express the identity of a subject, one has the choice between using the elements `<resourceRef>` if the subject is addressable, `<subjectIndicatorRef>` if the subject is non-addressable, and/or `<topicRef>`. The `<topicRef>` element is used to state that this topic has the same subject as another topic. Since the concept of a person is not addressable, we use `<subjectIndicatorRef>`. Further, we specify a name for the topic, using the `<baseName>` element. The `<baseNameString>` element provides the actual name for the topic. It is also possible to specify scope using the `<scope>` element and variant names using the `<variant>` element. Then we create a second topic representing the individual P25. To make the assertion that P25 is of type person, we simply use the `<instanceOf>` element and a link to the topic representing the concept of a person. The `<instanceOf>` element is how the type-instance relationship is expressed in XTM.

```
<topic id="person">
   <subjectIdentity>
       <subjectIndicatorRef
          xlink:href="http://www.ifi.uio.no/dmms/epicrisis_ex/person#Person" />
   </subjectIdentity>
   <baseName>
          <baseNameString>Person</baseNameString>
   </baseName>
</topic>

<topic id="P25">
    <instanceOf>
          <topicRef xlink:href="#person" />
    </instanceOf>
    <subjectIdentity>
       <subjectIndicatorRef
         xlink:href="http://www.ifi.uio.no/dmms/epicrisis_ex/individualEx#P25" />
        </subjectIdentity>
</topic>
```

Figure 10: A type-instance relationship in XTM 1.0

Next, we want to express that individual P25 has social security number "23098234125". Figure 11 shows how this is done in XTM. First we create a topic representing the concept of a social security number, which is the topic with id "hasPersonId". Then we extend the definition of the topic representing P25 by adding the `<occurrence>` element. The `<instanceOf>` element within the `<occurrence>` element states that this occurrence is of type "hasPersonId", and the `<resourceData>` element gives the value of the occurrence, namely the social security number.

```
<topic id="hasPersonId">
  <subjectIdentity>
     <subjectIndicatorRef
      xlink:href="http://www.ifi.uio.no/dmms/epicrisis_ex/person#hasPersonId" />
  </subjectIdentity>
  <baseName>
     <baseNameString>hasPersonId</baseNameString>
  </baseName>
</topic>

<topic id="P25">
  <instanceOf>
    <topicRef xlink:href="#person" />
  </instanceOf>
  <subjectIdentity>
      <subjectIndicatorRef
        xlink:href="http://www.ifi.uio.no/dmms/epicrisis_ex/individualEx#P25" />
  </subjectIdentity>
   <occurrence>
     <instanceOf>
      <topicRef xlink:href="#hasPersonId" />
        <resourceData>23098234125</resourceData>
     </instanceOf>
   </occurrence>
</topic>
```

Figure 11: How occurrence is expressed in XTM 1.0

Finally, we want to say that individual P25 is married to P26 and that P25 is the husband, and P26 is the wife. To do this, we must create topics representing P26 and the concepts of

married, husband and wife. Then we create an association using the <association> element. We specify what kind of association it is using the <instanceOf> element. The <member> element specifies the members of the association, P25 and P26. Within the <member> element, the <rolespec> elements states what kind of role the member plays in the association, meaning in our case whether it plays the role of the husband or the wife. The resulting XTM is displayed in Figure 12.

```
<topic id="married">
  <baseName>
      <baseNameString>Married</baseNameString>
   </baseName>
</topic>

<topic id="wife">
  <baseName>
      <baseNameString>Wife</baseNameString>
   </baseName>
</topic>

<topic id="husband">
  <baseName>
      <baseNameString>Husband</baseNameString>
   </baseName>
</topic>

<topic id="P26">
  <instanceOf>
    <topicRef xlink:href="#person" />
  </instanceOf>
  <baseName>
      <baseNameString>P26</baseNameString>
   </baseName>
</topic>

<association>
  <instanceOf>
    <topicRef xlink:href="#married" />
  </instanceOf>
  <member>
    <roleSpec>
      <topicRef xlink:href="#husband" />
    </roleSpec>
    <topicRef xlink:href="#P25" />
  </member>
  <member>
    <roleSpec>
      <topicRef xlink:href="#wife" />
    </roleSpec>
    <topicRef xlink:href="#P26" />
  </member>
</association>
```

Figure 12: Expressing an association in XTM

Notice in the examples above that the topic types, occurrence types and association types are not explicitly defined to be types when they are created. They are just regular topics until another topic, association or occurrence refers to it using the <instanceOf> element. So the topic map model can really evolve as it is being used.

4. Related Research

When starting work on this thesis, a lot of time was spent on trying to find research on using OWL on resource-limited devices. We were hoping to find research about or implementations of reasoners, query engines or other tools that could work with OWL on resource-limited devices, but instead we found articles stating that this was not possible, as mentioned in chapter 2. The SHARK project [41] had implemented a topic map engine specifically for resource-limited devices, so the focus shifted to instead look at how OWL could be translated into topic maps. It was only very late in my work, too late to look much into it, that I discovered Pocket KRHyper, a description logic reasoner for resource-limited devices.

4.1 Pocket KRHyper

Pocket KRHyper is explained in [42], [43], and [44]. It is open source and can be downloaded from http://sourceforge.net/projects/mobilereasoner.

Pocket KRHyper is a part of the IASON project by the working group Artificial Intelligence at Universität Koblenz-Landau. IASON stands for "A Location-Based Information Announcement System with Ontology-Based profiles". It is a project aimed at providing mobile users with location-aware personalized information. They define a semantic mobile environment consisting of service nodes and mobile users. The mobile users are devices such as PDAs, cell phones or smart phones. The service nodes are installed at points of interest, and broadcast semantically annotated messages to the mobile users that are within range. All communication is performed over the bluetooth protocol. The mobile users have profiles, and by performing matchmaking on the messages from the service nodes and the user profiles, the mobile users are able to filter out unwanted messages. This matchmaking is performed by description logic reasoning.

The IASON project does not want to be dependent of an internet connection to perform the matchmaking. This is because internet access from cell phones is expensive, and there are privacy issues with having to upload user profiles every time matchmaking is needed. All of the matchmaking has to be done on the user devices, which are limited in resources, so they created Pocket KRHyper, a description logic reasoner specifically for resource-limited devices. This reasoner is the first reasoner created for use on small mobile devices, and it seems that it is currently also the only one.

Pocket KRHyper is implemented in Java 2 Micro Edition (J2ME) and can be embedded in any Java application that uses J2ME, Java 2 Standard Edition (J2SE) or Java Enterprise Edition (J2EE). The reasoning tasks that are required for matchmaking are concept satisfiability and concept subsumption, and these are therefore the reasoning tasks that Pocket KRHyper can perform. Concept satisfiability tests whether a class can have any instances, and concept subsumption finds class hierarchies.

Pocket KRHyper has a description logic interface, but not a DIG or OWL interface. It provides support for a description logic called $\mathcal{ALCHI}$, which is a subset of the description logic underlying OWL Lite. In terms of OWL, it means that one can define classes using intersection, union and complement of other classes, as well as having universal and existential restrictions on properties [45]. Properties can be the inverse of other properties, and one can also have subproperties. This allows for pretty good expressivity. Even though Pocket KRHyper does not support OWL directly, is should be possible to translate OWL into

description logics since they are so closely related. Or one could look into implementing a DIG interface for the reasoner, which is mentioned as an “interesting thing to do” at the project website where you can download the reasoner.

4.2 SHARK

SHARK is an acronym that stands for “Mobile Shared Knowledge”. It is a project at the Technische Universität Berlin aimed at creating a distributed system for supporting organization, synchronization and exchange of knowledge for mobile users. Management of knowledge is handled by using topic maps and topic map engines. Exchange of information can occur both within specified user groups, and across different user groups.

SHARK assumes a network environment that contains both mobile and stationary devices, and is divided into the components Mobile Station, Central Station, and Local station. The Central Station is a server that stores the entire knowledge base, i.e. all of the topic maps, while the Mobile Station and Local Station are smaller devices and only store parts of the knowledge base. How the components are organized and interact with each other can be seen in Figure 13. SyncML is the protocol used for synchronization of knowledge, and KQML is the protocol used for knowledge exchange.

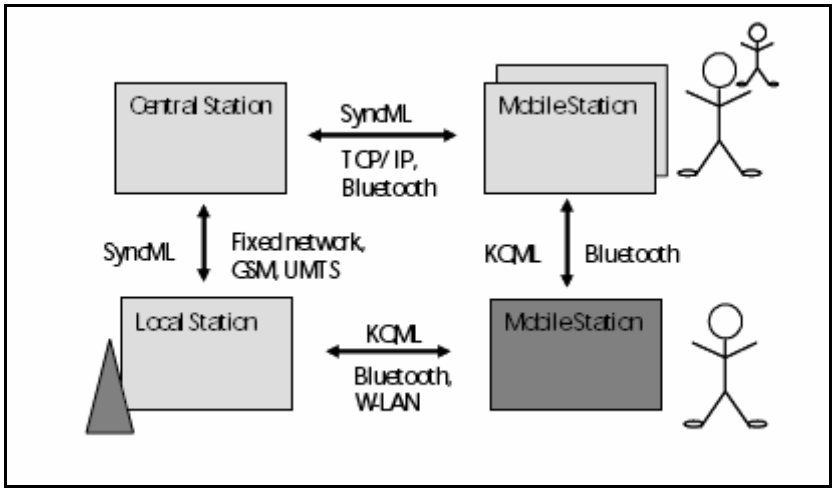

Figure 13: The different components of SHARK

What is interesting about SHARK, is that they have developed the first topic map engine that can run on a mobile device [1]. This engine is called MTV, which stands for “Mobile Topic Map Viewer”. It is implemented in J2ME and can run on any java enabled device. It is an implementation of the TMAPI interface, which will be explained further in section 6.2.4.3, but it is not a complete implementation as not all methods are implemented [2]. Reification is not implemented, and there is limited support for indexing. Vigdal [2] has done performance testing of MTV, and Andersen [1] has implemented merging functionality for the engine.

5. Examples of OWL ontologies in a rescue scenario

One of the main reasons for wanting to use OWL ontologies in the Ad-hoc InfoWare project, is there are already many existing ontologies that can be reused, so one does not have to create all new ones. However, there do not exist ontologies for everything, so some will have to be created for specific purposes. In this section I will give examples of some ontologies that may be used in a rescue scenario. These will just be examples, and the ontologies that would be used in a real-life scenario are likely to contain much more information. The ontologies are illustrated by their corresponding (partial) RDF graphs. In the graphs, all the nodes that start with "genid:" are simply anonymous nodes that have been given a generated id. The text in parentheses on some of the nodes specifies the types, and to make the graphs easier to read the XML qualified names are used instead of the full URIs. The RDF/XML for all of the ontologies can be found in Appendix A. They all use the sublanguage OWL DL and have been created using a tool called Protégé-OWL [46]. Protégé-OWL is an open-source ontology editor. It can load and save RDF/XML files, and uses the DIG interface to connect to reasoners.

In a rescue scenario, rescue personnel will be equipped with small mobile devices, and casualties may be equipped with different types of sensors to monitor their vital functions and position. Rescue personnel may receive updates on the status of the casualties by receiving readings from the sensors.

5.1 Person ontology (pers)

This ontology, shown in Figure 14, is a continuance from the previous example from section 3.3.1 where we convert the vocabulary `Person` defined in RDF to an ontology in OWL. An ontology about a person would normally contain more information, like the name, address, gender etc., but for the sake of the example I have only included the property `pers:hasPersonId`. The `owl:Class` element defines the class `pers:Person`. Then the `owl:Restriction` adds a restriction to the class for the property `pers:hasPersonId`, defined using the `owl:onProperty` property. `pers:hasPersonId` is declared to be both a functional and datatype property. The restriction uses an existensial quantifier for the property with filler being the datatype "positive integer". This restriction states that an individual of class `Person` (a person) must have at least one property of `pers:hasPersonId` with value a positive integer. Since this property is functional, a person can only use this property once. In other words, a person must have one, and only one, social security number. The restriction is contained inside the `rdfs:subClassOf` element. This is because classes are interpreted as sets, and the class `Person` is seen as a subset of all the classes that satisfy the restriction.

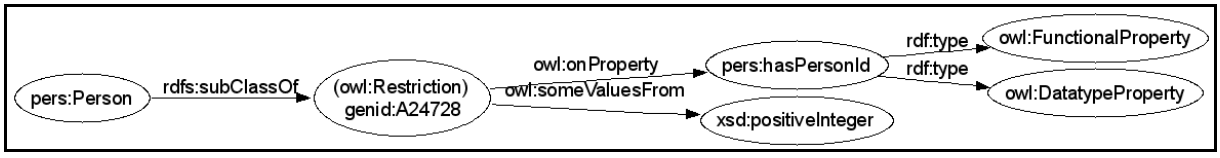

Figure 14: Person ontology

5.2 Epicrisis ontology (epi)

An epicrisis is defined in as "a secondary crisis in the course of a disease", but the word is also used to mean a synopsis of a patient's medical journal, giving enough information to be

able to follow up the patient. The `Epicrisis` ontology, seen in Figure 15, uses the latter meaning of the word. In this example I have only included person information for a target patient, which is defined by the property `epi:ofTargetPerson`. In this ontology, we will also use the `Person` ontology, so it is imported using the `owl:imports` property. An epicrisis contains information about exactly one person. Therefore we have created two restrictions on the class using the property `epi:ofTargetPerson`. All of the property values must be from the class `pers:Person`, and the cardinality is set to be exactly one. Instead of the last restriction, we could have made the property functional. But that could limit the reuse of the property in other ontologies, and in this case I think that the restriction belongs to the class, and not the property.

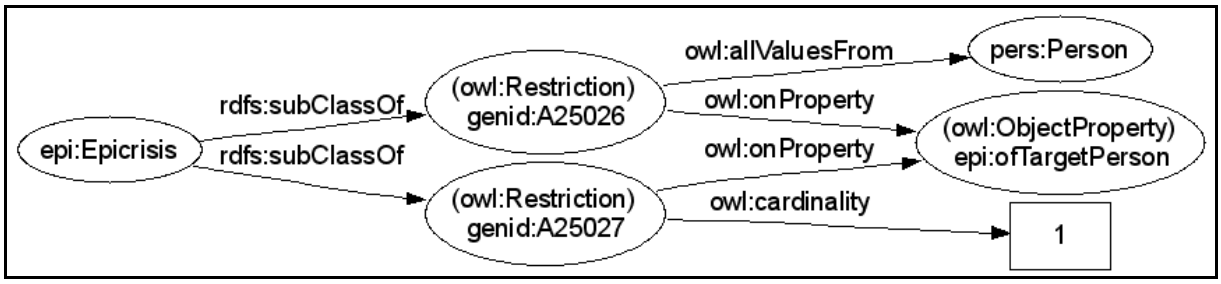

Figure 15: Epicrisis ontology

5.3 Alertness status ontology (cas)

This ontology is meant for defining the status of a patient. See Figure 16. Using this ontology, one can for instance find which patients are diseased and which patients require immediate attention. The class `cas:CasualtyStatus` is defined to be the union, using the `owl:unionOf` property, of the two disjoint classes `cas:CasualtyAlert` and `cas:CasualtyDiseased`, using the `owl:disjointWith` property. `cas:CasualtyAlert` defines the severity of a patient's status, and can be either one of "green", "yellow" or "red", defined by the property `owl:oneOf`. `cas:CasualtyDiseased` specifies that the patient is diseased, and consists only of the individual "diseased". At the end, the `owl:AllDifferent` element is used to state that all the individuals "green", "yellow", "red" and "diseased" are all different from each other. If they are not specifically stated to be different, OWL may under certain conditions assume that they are equal.

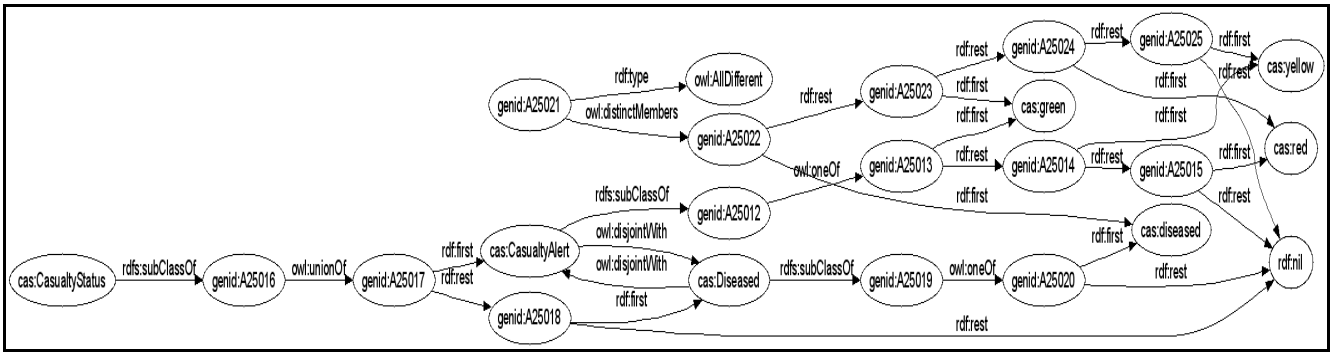

Figure 16: Alertness status ontology

5.4 Sensor Upper ontology (sns)

This ontology defines the common properties of the different types of sensors that are connected to a person. It defines the classes `sns:Sensor`, `sns:Observation`, and `sns:UnitOfMeasurement`. The `sns:Sensor` class is the superclass of the different types of sensors. It can have as subclasses for instance body sensors and GPS sensors. The `sns:Observation` class is the superclass of all the different types of observations reported from the sensors. The `sns:UnitOfMeasurement` class is used to define the type of the measurement from the observation. For instance the heart rate of a person is measured in beats per minute (bpm). An individual of the `sns:Sensor` class has a universal restriction on the property `sns:hasObservation` and on the property `sns:hasSensorId`. All observations have to be of the type `Observation`, and all sensor IDs have to be of type `xsd:String`. We have also added a cardinality restriction with value 1 on the property `sns:hasSensorId` to state that a sensor has to have one, and only one, ID. The `sns:Observation` class also has two universal restrictions. The `sns:atTime` property has to have a value from `xsd:dateTime`, and the `sns:withUnit` property has to have as a value an individual of the class `sns:UnitOfMeasurent`. See Figure 17.

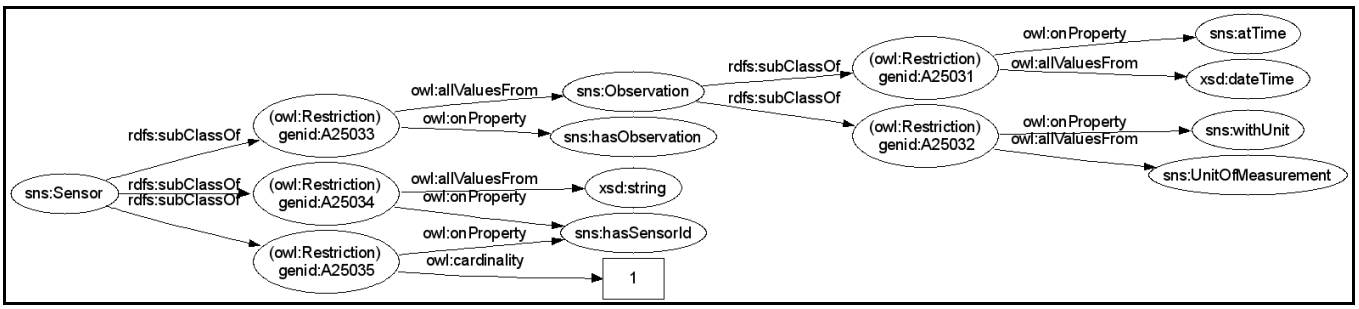

Figure 17: Sensor Upper ontology

5.5 Bodysensor ontology (bdsns)

This ontology defines the class `bdsns:BodySensor`, which is a subclass of `sns:Sensor`, and describes a sensor that is used for monitoring vital functions. Three types of body sensor observations are defined, `bdsns:HeartRateObservation`, `bdsns:BodytemperatureObservation`, and `bdsns:PulseOximetryObservation`. These observations all have restrictions on what kinds of values they may have. For instance, `bdsns:HeartRateObservation` can have value of type `xsd:positiveInteger` and unit `bdsns:bpm`. See Figure 18.

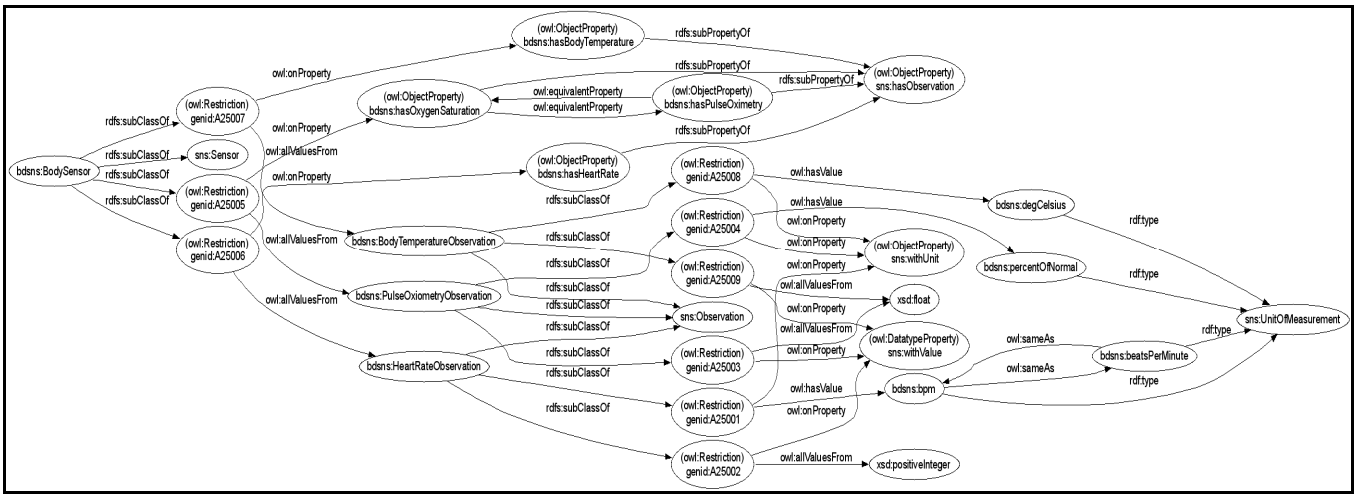

Figure 18: Bodysensor ontology

5.6 Rescue ontology (resc)

This ontology imports all of the previously defined ontologies. It displays how different ontologies can be "joined together" by relating the classes `epi:Epicrisis`, `cas:CasualtyStatus`, `sns:Sensor` and `pers:Person` to each other. First, it defines a class `resc:Person` that is defined to be equivalent to the class `pers:Person` in the Person ontology. This means that any member of the class `resc:Person` in this ontology, is a member of the class `pers:Person` in the Person ontology, and vice versa. They are basically the same class. Then the properties `resc:withCasualtyStatus`, `resc:withEpicrisis` and `resc:withSensor` are defined. All of these properties are used in separate universal restrictions that link the class `pers:Person` to the classes `cas:Casualty`, `epi:Epicrisis` and `sns:Sensor` respectively. See Figure 19.

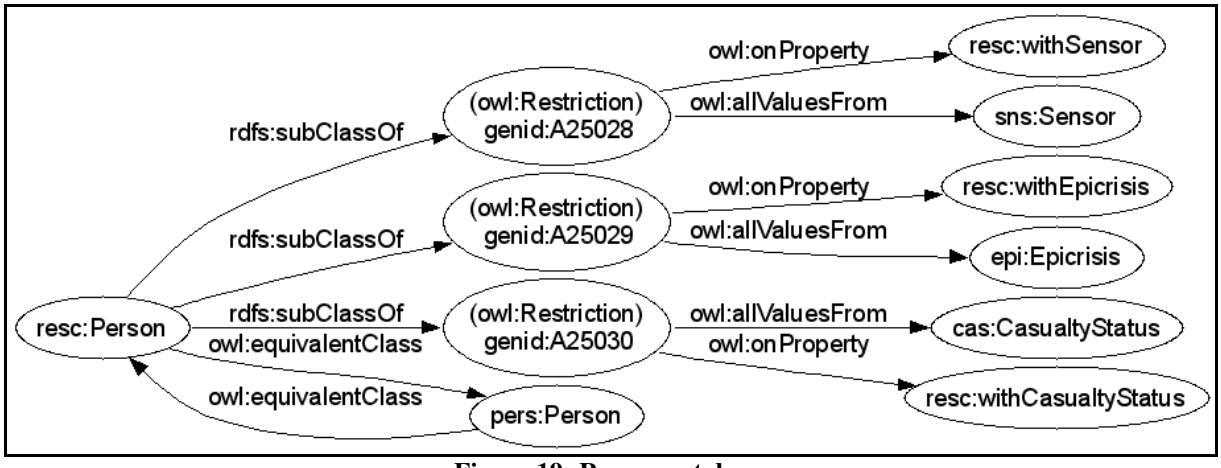

Figure 19: Rescue ontology

6. RDF/OWL vs. Topic Maps

Topic maps and RDF are very similar, and they try to solve the same problems of retrieving information in the electronic world. But their goals in how to achieve this, is different. Topic maps are used for creating a map over different resources to make it easier for people to find the information in those resources and connections between them. RDF is used for creating metadata about resources, and the metadata is structured in a way that one can use logical reasoning to find information. [47] gives a good picture of the differences between them. Topic maps are more suited for information management for humans. People are able to browse through the topic map and interpret the information that lies there. RDF and OWL, on the other hand, are better suited for interpretation by machines. Browsing through an OWL ontology is not easy for a person, but better for a machine, or rather an application, that can use reasoning engines to establish information about resources that is not inherently there. Even though they have these differences, they are currently being used in many of the same application areas, see Table 2 [48].

Application area	**Topic maps**	**RDF/OWL**
Findability	Yes	Yes
Portals	Yes	Yes
Content management	Yes	Yes
Enterprise Application Integration (EAI)	Yes	Yes
E-commerce	Yes	Yes
Knowledge Management	Yes	Yes
Semantic Web	Yes	Yes

Table 2: RDF and topic map application areas

One might wonder why there was created two different, but very similar, standards, where you have to choose between one or the other. It would seem to have been a lot easier just having one standard. The reasons are historical, and are explained in [49]. Topic maps and RDF were actually developed in parallel, without either one knowing much about the other. Work on topic maps was started in the early 1990s, and were adopted by ISO in 1996. The topic map standard was published in 2000, and the XTM 1.0 standard in 2001. Work on RDF was started by Guha at Apple. At that time, it was called the Meta Content Framework. The W3C continued the work, and published a recommendation in 1999, where they called it RDF. The two communities, ISO and W3C, did not become aware of each others work before in 1999, and then it was too late. If they had known about each other before, they might have cooperated to create one standard, or at least two standards that were capable of easy interoperability with each other. Now, both the standards are firmly established, and a merger at this point is considered to be impossible. Instead, the communities are looking at ways in which the two standards can interoperate. If it is possible to establish interoperability between the two standards, it will make it easier for applications to be able exchange data across them. This may make it easier for developers to decide more freely what standard to use, the standard that fits their needs the most, and not have to worry about whether the application will have to communicate with another application that is based on the other standard.

There has been created a very own group, the RDF/Topic Maps Interoperability Task Force (RDFTM) [50], for the purpose of creating guidelines for those who are interested in interoperability between the two standards. They have listed some short term and some long term objectives.

Short term objectives:

- Document strategies for representing topic maps using RDF/OWL and vice versa.
- Describe the pros and cons of existing approaches.
- Produce guidelines for transforming a topic map into an RDF/OWL representation and vice versa.
- Provide links to tools, applications, and papers on this topic.

Long term objectives:

- Proposing the guidelines described above for standardization in the W3C and ISO.
- Producing guidelines for using OWL to constrain topic maps.
- Producing guidelines for cross-querying RDF/OWL data and topic maps.

The short term objectives are in good progress with the publication of "A Survey of RDF/Topic Maps Interoperability Proposals" [51] and the draft "Guidelines for RDF/Topic Maps Interoperability" [52].

6.1 The main differences between RDF and Topic Maps

Garshol gives a thorough comparison between RDF and topic maps in [49], which this section will be based on. A comparison between OWL and topic maps is not included, as Garshol for the purpose of comparison regards OWL as a constraint language for RDF like TMCL will be for topic maps, although he acknowledges that OWL is actually a lot more than a constraint language. A correct comparison would therefore have to be between OWL and TMCL, but the latter is not completed, and a comparison is therefore currently impossible. Figure 20 shows how the different families of standards relate to one another.

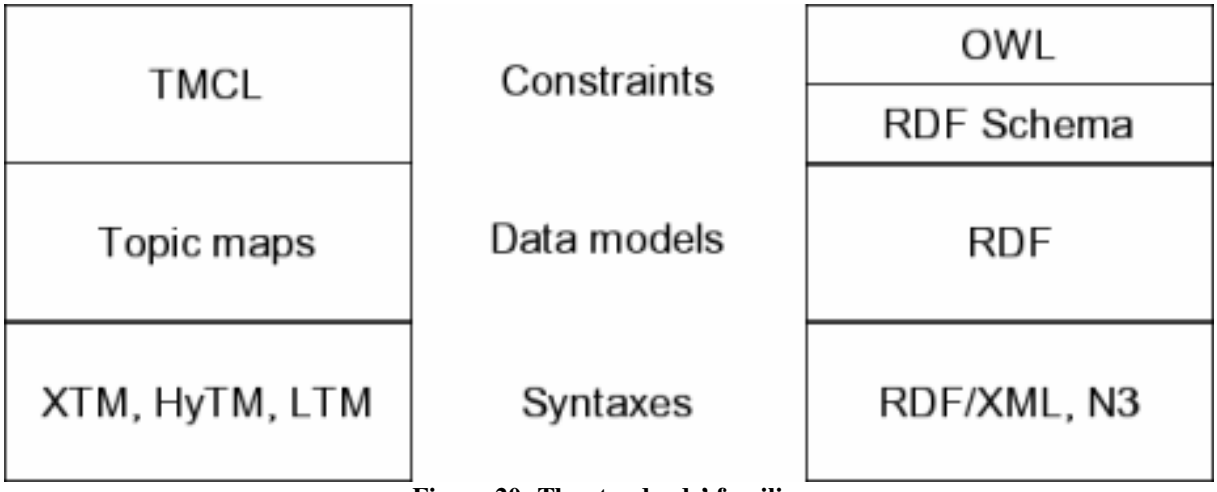

Figure 20: The standards' families

The comparison is based on the abstract models of the standards, the data models in Figure 20. For RDF, this means RDF graphs with nodes representing resources and arcs representing statements about the resources. For topic maps it means the concepts of a topic map which are topics, occurrences and associations.

Both topic maps and RDF can make representations of virtually anything whatsoever in this world. In RDF, "things" are called resources and are represented by nodes, and in topic maps the "things" are called subjects and are represented by topics. In both models, it is possible to say what type a "thing" is. In topic maps this is done using the type-instance relationship, and

in RDF it is done with a statement using the property `rdf:type`. A subtype-supertype association in topic maps can be used to create type hierarchies, and the same can be done in RDF using `rdfs:subClassOf`, although this is not really part of RDF itself but rather a part of RDFS.

Topic maps make a clear distinction between the actual real world element and its representation of it in the topic map by using the concepts of subject and topic, respectively. RDF, however, does not make such a distinction. This can be clearly seen in the way that the two standards use identifiers. In RDF, you only have one way to specify a URI identification of a resource. If the URI is a URL of a web page, for instance, you do not know whether the node using this URI is supposed to represent the resource itself or the information that is given by the resource. This distinction is made very clear in topic maps by the use of different ways to express identity using the concepts of subject locator and subject identifier. RDF models make use of three different types of nodes: URI nodes, literals, and anonymous nodes. A topic in a topic map can only be either a "URI topic" or an "anonymous topic" depending on whether or not it has been supplied with a subject identifier or locator. So even though the concepts of topics and nodes can be considered to be close, they cannot be considered equal.

When it comes to making assertions about "things", RDF has statements and topic maps have names, occurrences and associations. RDF is capable of making statements that correspond to names, occurrences and associations. But the statements are made using properties, and to know what these properties mean, one must know how to interpret the underlying vocabulary that declares the properties. In other words, one can easily make a statement supplying a name for a node, but then one must know that the property being used is supposed to be interpreted as a name.

Associations in topic maps are very different from a statement in RDF representing an association. In topic maps, associations do not have any directionality. Furthermore, associations can involve many topics, each having different roles to express how they are related to the association. RDF can only express binary relationships, and uses directionality. A note must be made here that if one uses OWL, one can decide that a property is symmetric, and therefore not have directionality, but this can not be done in RDF alone.

In topic maps, one is able to specify context for the different topic characteristics. RDF does not have anything similar built into the model. It is possible to use reification to achieve something similar to scope in RDF, but reification in RDF is not trivial.

Reification, is basically to make assertions about assertions. In topic maps, you can make an assertion about a name, occurrence or association by simply creating a new topic and giving it the subject identifier of the name, occurrence or association that you want to say something about. In RDF, it is a little more complicated as regular statements do not have an identity. In order to reify a statement in RDF, one must create a node of type `rdf:Statement`, and then add its subject, predicate and object through statements using the properties `rdf:subject`, `rdf:predicate` and `rdf:object`. As a result, the model is significantly changed as the path from the subject to the predicate to the object must be traversed through the new node of type `rdf:Statement`. This is a recognized problem, but reification is not very often used, and by not supporting it directly, like topic maps do, makes RDF more light-weight.

Based on the differences mentioned in this section, Garshol concludes that RDF is a more light-weight model than topic maps, and that topic maps is a higher-level model than RDF.

This is because topic maps have more built into it than RDF. Further, a merging of the two standards is impossible without having to make big changes in one of them, changes that would most likely be unacceptable to their users. Instead, one should concentrate on methods to convert data between the two standards.

6.2 Proposals for translation between RDF and Topic Maps

In [51], a survey is conducted of the different existing proposals for translating between topic maps and RDF. It is clearly stated that the goal of the current work being done in the area of interoperability is only related to RDF and topic maps, and not RDFS and OWL, even though these are extensions of RDF. The proposals that are evaluated in this survey were chosen based on how complete and well-documented they are. This was important to be able to examine them more closely.

The goal is to achieve interoperability at the data level only. This gives three requirements:

1. It should be possible to transfer data between the two models without unacceptable loss of data.
2. It should be possible to query the result of a translation in terms of the target model.
3. It should be possible to share vocabularies across the two models.

The different proposals are evaluated using two criteria, the completeness of the proposal, and the naturalness. Completeness is defined as the ability of the translation to map every semantic construct in the source model to the target model without loss of information. If a translation is complete, it will also be reversible. The naturalness of the translation is evaluated based on whether the translation from the source model to the target model yields a similar result to how the same information would have been expressed in the target model in the first place. To decide on the naturalness of each translation proposal, they have created a test case where the same information is expressed in both RDF and topic maps, so that when a translation is performed from RDF to topic maps, the result can be compared to how the same information originally would have been expressed in topic maps, and vice versa.

There are mainly two ways to perform a translation between RDF and topic maps, “mapping the model”, which is a semantic mapping, and “modelling the model”, which is an object mapping. When you perform a semantic mapping, you try to find equivalent concepts in the source model and the target model and map between these. As we have seen earlier, there are many similar concepts in RDF and topic maps, but they are not similar enough to provide a complete generic mapping between them. An object mapping, on the other hand, uses the concepts in one model to describe the other model.

There are five main proposals that were chosen to be in the survey. They are named based on their author, or in the case of multiple authors, their affiliation. Many of them were written before the two standards were formalized, which explains why they are incomplete. Table 3 gives a summary of the different translation proposals. The columns “RDF2TM” and “TM2RDF” state whether the proposal has a translation from RDF to topic maps or from topic maps to RDF, respectively. An “X” gives a positive indication. The “Type of mapping” column specifies what kind of mapping is used, if it is an object mapping or a semantic mapping. In the proposal by Moore, there are suggestions for both object and semantic mapping, so these have been separated in the table. The “Completeness” column implies how complete the models are. Since one cannot really measure completeness, the table just gives

an indication, and the completeness of the proposals will be further elaborated. The "Implemented" column states whether the translation has been implemented or not.

Proposal	RDF2TM	TM2RDF	Type of mapping	Completeness	Implemented
Moore - object mapping	X	X	Object	Fairly complete	
Moore - semantic mapping	X	X	Semantic	Incomplete	
Stanford		X	Object	Incomplete	X
Ogievetsky		X	Object	Complete	X
Garshol	X	X	Semantic	Almost complete	X
Unibo	X	X	Hybrid	Fairly complete	X

Table 3: Summary of translation proposals

6.2.1 Moore's proposal

The first proposal written on interoperability between RDF and topic maps, was by Graham Moore [53]. He looked both at semantic mapping and object mapping, but concluded that a semantic mapping was to be preferred because it gave a more natural result. When using an object mapping, the result is expressed in terms of the other model, which is very unnatural, and it is not possible to query the result in a "normal" way. Also, an example is given for object mapping where a binary association that consists of five topics is mapped to twenty-two statements in RDF. When using semantic mapping, the same association only needs two RDF statements. This obviously shows that the results of an object mapping gives a lot more "bloating" than a semantic mapping.

For his object mapping proposal, he defines ways to map from topic maps to RDF and from RDF to topic maps, but neither of them are reversible. In other words, it is only possible to be working in one domain or the other. When going from RDF to topic maps, he suggests defining PSIs for every construct in RDF, and then expressing RDF statements as ternary associations using role types called "rdf-subject", "rdf-property", and "rdf-object". In topic maps, literals can not be a member in an association, but they can often be the object of an RDF statement. To represent a literal in an association would require it to have an identifier, which literals in RDF don't usually have. Moore states that he is aware of this problem, but does not give any suggestions for a solution, which makes the proposal not entirely complete. Mapping from topic maps to RDF is similar to going to the other way. RDF properties are defined for every topic map construct.

Moore's proposal for a semantic mapping is very incomplete, [51] describes it as only a sketch that focuses on differences between RDF statements and topic map associations. The main problem is that an RDF statement only contains three pieces of information (the subject, object and predicate), while an association in a topic map contains five pieces of information (two members, two role-types, and an association type).

6.2.2 The Stanford Proposal

This proposal only allows for a translation from topic maps to RDF, and uses an object mapping approach. They use a topic map model called "Processing Model for Topic Maps" (PMTM4). This is a model that did not gain much popularity, and has been superseded by the TMDM. The proposal is regarded as complete with respect to the PMTM4, but since this model does not handle the use of URIs and strings, it is not a complete proposal with regards

to topic maps. As Moore's proposal for an object mapping, translation using this proposal also results in a lot of "bloating".

6.2.3 Ogievetsky's proposal

The Ogievetsky proposal only specifies how to translate from topic maps to RDF. It is basically a proposal on how to translate XTM to RDF, and is therefore very syntax oriented. It uses an RDF vocabulary called RTM that consists of eleven classes and seventeen properties that all represent topic map concepts. The translation is done through an XSLT transformation, and maps the topic map constructs in the XTM file to the corresponding classes and properties defined in the RTM vocabulary. The resulting RDF/XML from the transformation looks very similar syntactically to how it would be expressed in XTM. The proposal is complete in that it covers every aspect of XTM syntax, but it scores low on naturalness. This is because an association that usually would have been expressed using only one RDF statement takes seven RDF statements after a translation using this proposal.

6.2.4 Garshol's proposal

Garshol [49] recognizes two obvious ways in which RDF and topic maps can interoperate. The first way is to do traditional conversion of data from one model to the other. The second way is that they can be used together in an application where topic map data can be seen as RDF data and vice versa through a particular interface.

Garshol has chosen a semantic mapping approach in his proposal. He has earlier written an object mapping proposal, but he abandons this approach as it is "heavyweight and awkward to work with". When using object mapping for translating topic maps to RDF, the result will not interoperate cleanly with other RDF data, and therefore one can not query the data in the same way.

This proposal is the only one that makes suggestions for how RDFS and OWL also can interoperate with topic maps.

6.2.4.1 Mapping from RDF to Topic Maps

A generic semantic mapping is considered to be impossible. This is because the RDF model does not contain enough information to be correctly mapped to a topic map. For instance, if you have an RDF statement where the object is a literal, there is no way of knowing if this statement should be mapped to a name or an occurrence without knowledge about the property being used. Similarly, if you have an RDF statement where the object is a URI, one does not know whether to translate it to an association or an occurrence in topic maps. The only way for the computer to know how to perform the mapping, is if there is information available about what the RDF properties that are used actually mean. Therefore, Garshol suggests the use of an RDF mapping vocabulary called RTM [54] (not the same RTM vocabulary suggested in Ogievetsky's proposal).

When mapping an RDF statement to topic maps, the subject of the statement can easily be mapped to a topic, but the problem is how to map the object, or to figure out the intended meaning of the predicate. Table 4 gives an overview over what the object of a statement can be mapped to and what additional information is required to represent it in a topic map.

Mapping to	Information needed	Legal node types
Name	Scope	Literal
Occurrence	Type and scope	Literal and URI
Association	Type, scope, and role types	URI and blank
Subject address		URI
Subject identifier		URI

Table 4: Information required and possible mappings to topic map constructs for objects in an RDF statement

Using the RTM vocabulary, one can specify which one of the possible mappings in Table 4 fits best with the intended meaning of the properties in an RDF vocabulary. An advantage to using a mapping vocabulary in RDF is that every RDF vocabulary can express its own mapping information in the same file. The vocabulary consists of five properties and seven classes, which are explained below (`rtm` is the namespace for http://psi.ontopia.net/rdf2tm/#).

- `rtm:maps-to` This property is used to say what an RDF property is to be mapped to in a topic map. All properties that are used in an RDF vocabulary that is being mapped to topic maps must specify a mapping using this property. If no mapping is specified, the original RDF property will just be ignored during the translation. The possible values for the object of this property is one of the following classes:
  - `rtm:basename` The property is mapped to a base name.
  - `rtm:occurrence` The property is mapped to an occurrence.
  - `rtm:association` The property is mapped to an association.
  - `rtm:instance-of` The property is mapped to a type-instance relationship.
  - `rtm:subject-identifier` The property is mapped to a subject identifier.
  - `rtm:subject-locator` The property is mapped to a subject locator
  - `rtm:source-locator` The property is mapped to a source locator.
- `rtm:type` This property is used to specify the type of an occurrence or association. The default type is the RDF property itself.
- `rtm:in-scope` This property is used to specify the scope for topic characteristics.
- `rtm:subject-role` This property specifies a role for the subject of an RDF statement where the predicate is mapped to an association.
- `rtm:object-role` This property specifies a role for the object of an RDF statement where the predicate is mapped to an association.

For example, if we wanted to state that the property `pers:hasPersonId` should be mapped to an occurrence, we could include the mapping information shown in Figure 21.

```
<rdf:Description
  rdf:about="http://www.ifi.uio.no/dmms/epicrisis_ex/person#hasPersonId">
 <rtm:maps-to rdf:resource="http://psi.ontopia.net/rdf2tm/#occurrence"/>
</rdf:Description>
```

Figure 21: Expressing that an RDF property should be mapped to an occurrence in topic maps.

Mapping from RDFS to Topic Maps

RDFS can be viewed as vocabulary for RDF, and therefore be mapped in the same way as RDF. Classes in RDFS are simply mapped to topics, and one can specify how properties should be mapped using the RTM vocabulary.

The equivalent to the `rdfs:subClassOf` property, is the supertype-subtype association in topic maps. The `rdfs:label` property is defined in [17] as a "human-readable version of a resource's name". This definition fits well with the definition of a base name in topic maps, so these two may be considered to be equivalent. This does not mean that other properties defined in RDF can not represent names and be translated into base names in topic maps. `rdfs:comment` is a property that can be used to give a description of a resource, which fits well with an occurrence in topic maps. This mapping information can be expressed using the RTM vocabulary as shown in the RDF graph in Figure 22.

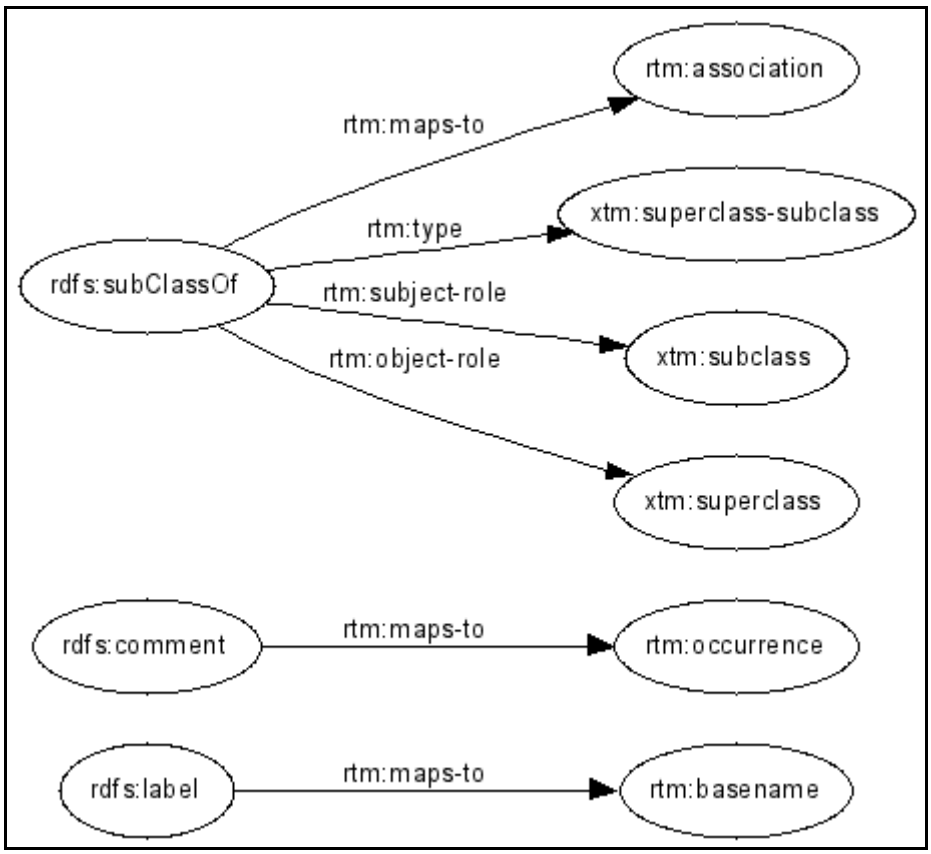

Figure 22: Mappings of RDFS properties to topic maps

Garshol also explains how RDFS can be mapped to a schema language for topic maps called the The Ontopia Schema Language (OSL). This is a language where one can state simple constraints for topic maps that can be used in the absence of the topic map constraint language

TMCL. [55] says that TMCL will be similar to OSL, but more powerful and probably use a different syntax. For this reason, and the fact that OSL is not part of topic maps itself, I will not go into details about OSL here.

Mapping from OWL to Topic Maps

Same as with RDFS, OWL can be viewed as a vocabulary for RDF, and can therefore be mapped to topic maps in the same way as RDF.

For all metadata information about OWL ontologies, he suggests mapping them to topic maps directly as descriptive properties. This includes the properties `owl:imports` for importing other ontologies, `owl:versionInfo` to say what version an ontology is, `owl:priorVersion` where the object is a previous version of this ontology, and `owl:backwardCompatibleWith` and `owl:incompatibleWith` which states whether or not the ontology is compatible with other versions.

The properties `owl:equivalentClass` and `owl:equivalentProperty` are used to specify that classes or properties have the same extension, but not the same intension. There is nothing equivalent to this in topic maps, so Garshol suggests that they are mapped to associations. The same goes for the property `owl:disjointWith` which is used to state that an instance of one class can not also be an instance of the other class.

In OWL, one can state that an object property is the inverse of another property using `owl:inverseOf`. Garshol says that since object properties are mapped to associations in topic maps, this property is not needed since associations do not have directionality.

His suggestions expressed using the RTM vocabulary can be seen in Figure 23, shown as an RDF graph.

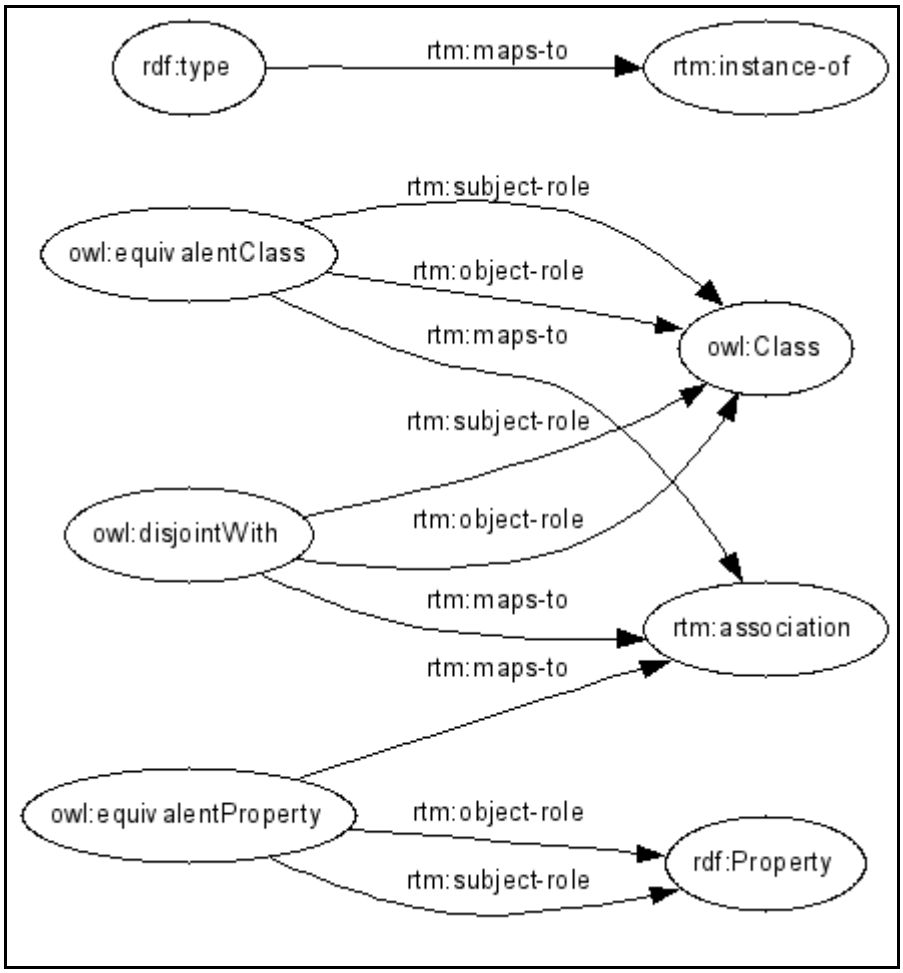

Figure 23: Mappings of OWL properties to topic maps

6.2.4.2 Mapping from Topic Maps to RDF

When translating the other way, from topic maps to RDF, Garshol identifies some issues that are problematic for a generic translation. Names in RDF can be expressed by many different properties, and it is therefore not possible to know what property to map a base name to. Associations can be translated into a statement where the predicate is the association type, but one does not know which role in the association is the subject and which is the object. In a case where a topic has more than one identifier, one has to pick which one to use as the URI in RDF. He makes some suggestions for solutions to these and other issues regarding translation from topic maps to RDF, which are listed below.

- Multiple URIs for the same topic can be handled using the RDF properties for equivalence found in OWL.
- Associations with more than two roles can be turned into resources whose type is the association type, and each role can then be represented as a separate statement with the role type as the property and the association resource as the subject.
- Reification and scoping can in general be represented by using RDF reification to represent the statement that would connect the topic characteristic with the topic. A special property will have to be defined for representing scope. As for the reification this is done by simply merging the resource representing the topic characteristic assignment with that representing the reifying topic.

- Binary non-symmetric associations can be handled by having the mapping contain one association from the association type to the preferred subject role.
- Selection of name properties can be done by having the mapping contain an association from the topic type to a topic representing the preferred RDF name property.

In addition to these suggestions, a topic map vocabulary called TMR [56] has been created where one can define specific mappings, just like one can with RTM. `tmr:name-property` is an association type that is used to specify what RDF property the base name of a topic type should be mapped to. The `tmr:preferred-role` association type is used to specify the directionality of the RDF property that a binary association will be translated into.

The survey regards Garshol's proposal as being almost complete, and that it gives a natural result translating in both directions.

6.2.4.3 Implementations

The proposal by Garshol has been implemented more or less in two applications, in the library TMAPI-utils [57] which is an addition to the Common Topic Map Application Programming Interface (TMAPI) [58], and in the Omnigator which is a part of the Ontopia Knowledge Suite created by Ontopia. Each of these implementations will be explained next.

TMAPI-utils

TMAPI is a programming interface written in Java. It defines a set of core interfaces that should be a part of a topic map application. There are several implementations of the TMAPI interface, maybe the two most commonly known are tinyTIM (tiny Topic Map Engine) [59] and TM4J (Topic Maps For Java) [60]. TMAPI, TMAPI-utils, tinyTIM and TM4J are all open source software and freely available to download.

TMAPI-utils are a set of tools that can be used with the TMAPI interface implementations. It includes classes for serialization and deserialization of topic maps. It also includes a partial implementation of Garshol's proposal for mapping from RDF to topic maps using the RTM vocabulary. This implementation is not a part of the release so the code must be checked out from the repository. This suggests that it is not completely finished yet, and explains why the implementation is not complete. The implementation is only partial because it does not handle translations of RDF properties to topic map associations. In addition, it requires every single property in the RDF vocabulary that is being mapped to have mapping information. There is in other words no default mapping, and properties that do not have mapping information, are ignored.

Omnigator

The Omnigator is available for download from http://www.ontopia.net as a part of the package OKS-samplers. The download is free in that you do not have to pay for it, but it is not open source, and it does come with a license that limits its use. The tools included in the OKS-samplers package are not all full-version and are intended for evaluation purposes only. It is also possible to purchase a full version of the Ontopia Knowledge Suite.

The Omnigator is a tool for browsing, querying and merging anything that can be considered to be a topic map. It can take as input the formats XTM, HyTM, LTM and RDF, and can also export topic maps to these formats. When using RDF as input, you can specify mapping information using the RTM vocabulary in the RDF file itself, or in an own file containing all the mappings used by the Omnigator. If there is no mapping information available, it makes a

default mapping proposal based on a “best guess”. For instance, a RDF statement where the object is a resource is proposed mapped to an association with role types `rtm:subject` and `rtm:object` for the subject and object of the statement. If the statement has a literal as an object, it is proposed mapped to an occurrence. It is possible to change how properties are mapped in the Omnigator itself.

Included in the OKS-samplers are the Vizigator and Ontopoly. Vizigator gives you a graphical view of a topic map which can be browsed, and Ontopoly is a tool for creating and editing topic maps.

6.2.5 The Unibo proposal

The authors of this proposal rejected the pure object mapping approach because it gives a much more unnatural result than a semantic mapping does. The Unibo proposal is much more similar to Garshol’s in that they offer a generic mapping that can be improved with more specific mapping information that can be expressed in an XML vocabulary. This vocabulary allows you to express how a binary association in topic maps should be mapped to statements in RDF, and whether a RDF property should be mapped to an association or an occurrence. The main difference between this proposal and Garshol’s is that this proposal makes use of more properties inherent in RDF and RDFS for the generic mapping, and that an XML vocabulary is used instead of an RDF vocabulary. Also, in some cases where a generic mapping is not completely possible, the proposal makes use of an RDF vocabulary, tm2rdf, for modelling topic map concepts. So this proposal is actually a hybrid solution since it makes use of both semantic and object mapping.

Subject locators in topic maps are equated with resource URIs in RDF, while subject indicators are represented by `rdfs:isDefinedBy`. This will work well when going from topic maps to RDF, but going the other way it may cause resources that are clearly non-addressable to be translated into addressable subjects. This is because in RDF you cannot assume that the resource URI is an addressable resource. Topics that do not have a subject locator are mapped to a blank node in RDF, and the base name of the topic is used to generate an id for the blank node. When going the other way, the id of a blank node is used to generate a base name for the topic that the node is mapped to. A base name in topic maps is mapped to the `rdfs:label` property in RDF, and vice versa. For handling names that have variants, the proposal suggests using the property `tm2rdf:baseName` where the object is a blank node with properties `rdfs:label` for the base name and `tm2rdf:variant` to express the variant names. The object of the `tm2rdf:variant` property is another blank node that has properties `tm2rdf:variantName` and `tm2rdf:parameter`. Figure 24 shows how the RDF graph would look like when mapping a base name with a variant name from a topic map to RDF. Scope is handled using the property `tm2rdf:scope`.

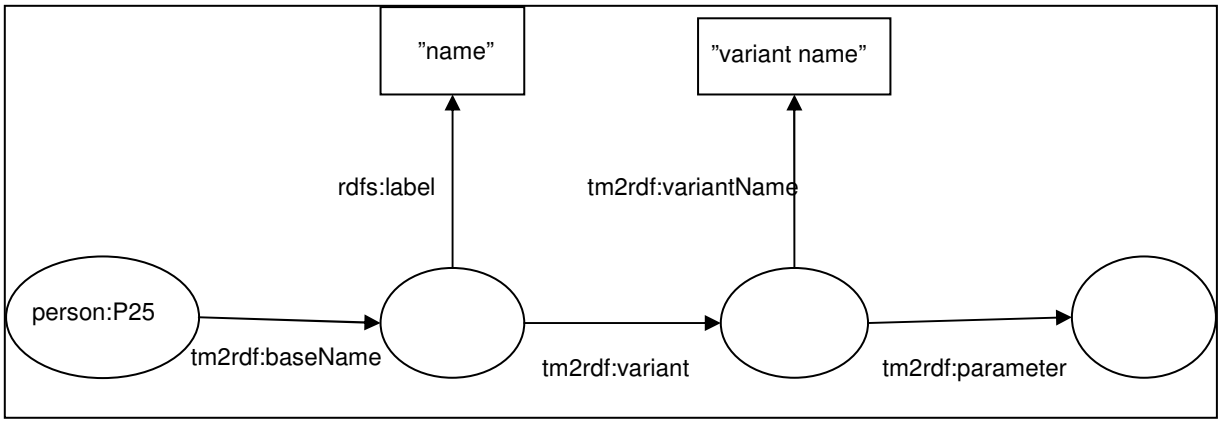

Figure 24: RDF graph from translation of base name with variant name from a topic map to RDF.

The way that identity is handled makes it look like this proposal is best suited for doing a mapping from topic maps to RDF, and then back to topic maps again. It is not very well suited for a mapping where the original model is in RDF.

The proposal suggests two alternatives for translating associations in topic maps to RDF. Each of the alternatives has variations. When using one of the alternatives, it results in losing either information about role types or association types, so the other alternative is preferred. The other alternative uses a statement where the predicative is the property `tm2rdf:association` and the object is an RDF bag of blank nodes that each represents a member in the association. When going from RDF to topic maps, the default is to map an RDF statement to an association, but this mapping can be controlled by the user through using the XML vocabulary.

Further, the `rdf:type` property is equalized to the type-instance relationship in topic maps. The `rdfs:subClassOf` property is mapped to the supertype-subtype association in topic maps, a specialized association that uses PSIs for the role types and the association type.

The survey regards this proposal as fairly complete and with some reversibility, but when performing a roundtrip, the result may not be the same as the starting point. It further states that translations where additional mapping information is supplied using the XML vocabulary gives a much more natural result than the default translation. The results from using the test cases show that when using the default mappings, going from topic maps to RDF results in thirty-eight statements to represent what would normally only need twelve statements. This result may have been improved if one had added additional mapping information. Results from going the other way are not provided. This is because at the time the survey was written, the implementation of this proposal going from RDF to topic maps seemed to be in error.

6.2.6 Discussion

Based on the results of the survey, a semantic mapping seems to be preferable to an object mapping because it produces a much more natural result. When the result is natural, one can treat topic map data as RDF data and vice versa. This makes it easier for querying since you do not have to take into account what the originating model was. On the other hand, a generic mapping does not seem to be possible, so a semantic mapping requires extra mapping information to be supplied with every vocabulary. With an object mapping a generic solution is possible, but the result does not interoperate cleanly with other data and one has to use special queries for data that has been converted. Also, an object mapping produces a lot of

“bloating”. When mapping from topic maps to RDF, for instance, a lot more statements are produced than what would be needed if the same information had been originally written in RDF.

Out of the five proposals that have been described, only two of them are pure semantic mappings. These are Moore’s and Garshol’s proposals. Since Moore’s proposal is very incomplete, Garshol’s seems to be the best choice available. He also has included suggestions for how to convert from RDFS and OWL to topic maps.

7. OWL, Topic Maps and the Ad-hoc InfoWare Project

So far, we have looked at OWL and topic maps, compared them, and seen proposals for how one can translate between them. Now, let's look at how this can be applied to the Ad-hoc InfoWare Project.

One of the strengths of OWL is its support for reasoning, something topic maps do not have. Still, in this thesis we are looking at how OWL can be translated into topic maps, and then losing the ability to reason over the ontology. We are not saying that reasoning is suddenly not important, but some of it can be performed before the translation from OWL to topic maps, and therefore the information inferred from the reasoner can also be translated to the topic map. A topic map engine can then be used for browsing and querying the ontology on a resource-limited device. It may seem cumbersome to have to translate the ontology from OWL to topic maps for only to be able to browse and query the ontology. It would then seem that a simple query engine for OWL would suffice. Unfortunately, we have not been able to find such tools for OWL that are designed for resource-limited devices.

The problem with using OWL in a MANET is that reasoning is a very resource consuming task, and therefore will be problematic on small resource-limited devices. I would imagine that the kind of reasoning that would most likely be needed in a rescue situation is support for concept satisfiability testing, consistency checking, and subsumption testing. Concept satisfiability testing is important to make sure that classes are not defined using contradictory statements, and consistency checking is important to ensure that inconsistencies are not created when adding data. Inconsistencies can come from trying to add the same individual to disjoint classes or not upholding the restrictions that are defined for a class. Subsumption testing can test whether a class is the subclass of another class. This can be useful if one is interested in retrieving all the individuals of a class, and therefore also the individuals of all of the subclasses of that class. I do not think instance checking is of too much importance since I am assuming that all data that is added during a rescue operation are created as instances of the defined classes, and subsumption testing can be used to find whether an individual is an instance of a class by following the class hierarchy.

In section 3.2 I mentioned the different phases of a rescue scenario. All ontologies are assumed to be created in the a priori phase, which can be done on a resourceful machine. Once an ontology is created, one can perform subsumption reasoning on the ontology, and add all of the information that is inferred from this reasoning to the ontology directly. This means that if the reasoner discovers subclass-superclass relationships that are not explicitly stated in the ontology, these relationships can be added to the ontology and become explicit statements. This reasoning can also be performed on a stronger machine in the a priori phase, and may therefore not be necessary to perform during the rescue operation, assuming that it is not allowed to create new classes during an operation but only add instances to classes. This is a logical assumption because users are generally not given the opportunity of modifying the data model of an application that they are using. An ontology is a data model, and can also be compared to a database schema which is not supposed to be changed while in use. If the ontology was allowed to change during use, there could be created inconsistencies in the model, which is not desirable. In such a case, it would be required to have a reasoner on the resource-limited devices to make sure that inconsistencies were not created and to infer new knowledge from the model as it changed.

It is also possible to perform concept satisfiability testing of an ontology in the a priori phase of a rescue scenario, making sure that it is possible for classes to have any instances. However, consistency checking is the one reasoning task where it does not suffice to perform the reasoning before an operation. This is because data that can cause inconsistencies may be added during the rescue operation. When an ontology defined in OWL is translated into a topic map, regular topic map tools are not able to ensure that inconsistencies are not created when adding new instances to the model. In other words, one can not make sure that restrictions are upheld unless one has an application that is designed for this purpose. This is discussed more in chapter 9.

The translation from OWL to topic maps can also be done in the a priori phase. Therefore one does not need to worry about resource constraints for the application performing the translation.

8. Using the RTM vocabulary for mapping between OWL and Topic Maps

Garshol's proposal, which was described in section 6.2.4, makes suggestions for how some properties in RDFS and OWL can be mapped to topic maps, but he does not cover all of the properties. For the remaining properties one could choose to use the default mappings suggested by the Omnigator. This means that RDF statements that have URIs as the object will be mapped to associations where the subject gets the role of `rtm:subject`, and the object gets the role of `rtm:object`. URIs in RDF are always mapped to subject identifiers in topic maps. This makes sense because you do not know whether a URI in RDF represents an addressable or a non-addressable resource. So to be sure that non-addressable resources are not mapped to addressable subjects in topic maps, it is the safest choice to translate URIs to subject identifiers. Statements that have literals as values will be mapped to occurrences. There may however be some more improvements that can be made to the default mappings. Some properties may be mapped to other topic map constructs to make the model fit better with the topic map paradigm. Also, readability of the resulting topic map may be improved by using other role types in associations than the default ones. One can look to the domain and range of a property for suggestions for role types. The domain and range of a property specifies what classes are allowed to be the subject and object of the property, and also classifies individuals to be members of classes. So one could say that the domain and range of a property say something about the roles of resources in a statement. This will not work in all cases, though, since some properties have the same class as both range and domain, and in order to see the directionality of the property, it is better to use the default values.

8.1 From RDF to Topic Maps

First, let us look at how the built-in properties of RDF can be translated into topic maps. The simplest is the `rdf:type` property because it can be translated directly into a type-instance relationship in topic maps.

In RDF you can give a statement an identifier by creating an instance of the class `rdf:Statement`, and using the properties `rdf:subject`, `rdf:predicate`, and `rdf:object` for expressing the subject, predicate and object of the statement. Since these properties are used for associating resources, they are most naturally mapped to associations in topic maps. These three properties all have `rdf:Statement` as domain and `rdf:Resource` as range. One can say that an instance of the class `rdf:Statement` also plays the role of `rdf:Statement` and that the subject, object and predicate are resources that are linked to the statement. It therefore makes sense to use the domain and range of the properties as the association role types in topic maps.

The properties `rdf:first` and `rdf:rest` are used for expressing collections in RDF. Topic maps do not have an equivalent to these properties, and since they link resources together, they can be translated into associations. A collection in RDF is like a linked list of anonymous nodes which each has a pointer to a resource using the `rdf:first` property, and a pointer to the next element in the list using the `rdf:rest` property. The `rdf:first` property has `rdf:List` as domain and `rdfs:Resource` as range, which will fit well as association role types in topic maps. The `rdf:rest` property has `rdf:Resource` as both its domain and range. Since this is a property that clearly expresses some directionality, showing which

element is next in the list, it is probably better to use the default association role types when translating to topic maps.

The `rdf:value` property does not have an equivalent in topic maps, and since the object of this property can be a literal, it is best mapped to an occurrence in topic maps. Table 5 gives a summery of how the RDF properties can be mapped to topic maps.

Property	Mapped to	Association type	Subject-role in association	Object-role in association
rdf:type	type-instance relationship			
rdf:subject	Association	rdf:subject	rdf:Statement	rdfs:Resource
rdf:predicate	Association	rdf:predicate	rdf:Statement	rdfs:Resource
rdf:object	Association	rdf:object	rdf:Statement	rdfs:Resource
rdf:value	Occurrence			
rdf:first	Association	rdf:first	rdf:List	rdfs:Resource
rdf:rest	Association	rdf:rest	rtm:subject	rtm:object

Table 5: How RDF properties can be mapped to topic maps

8.2 From RDFS to Topic Maps

Now, let us look at how RDFS can be translated into topic maps. Translating classes are easy, a topic is created for every class. RDFS properties, on the other hand, are a little trickier.

The properties that Garshol does not make mapping suggestions for are `rdfs:domain`, `rdfs:range`, `rdfs:seeAlso`, `rdfs:isDefinedBy` , `rdfs:member`, and `rdfs:subPropertyOf`. Topic maps do not have anything similar for representing the domain and the range of a property. This information is important in RDFS, and instead of it being lost in a translation, the properties can be translated into associations with the corresponding association types. Also, both `rdfs:range` and `rdfs:domain` have range `rdfs:Class` and domain `rdfs:Property`. Statements using the properties `rdfs:range` and `rdfs:domain` can then be mapped to associations where the subject gets the association role type `rdfs:Class`, the object gets the association role type `rdfs:Property` and the predicate is the association type. Using these role types can work because one can say that a class is the domain of a property, and that a property has the class as a domain.

The property `rdfs:seeAlso` can be viewed as similar to a *see also* reference in an index, which in topic maps is represented as an association between two topics. This property is therefore best translated into an association in topic maps. The association roles in this association do not need to represent any directionality because a *see also* reference can usually go both ways between resources. Both the domain and range of this property is `rdfs:Resource`, which becomes the role type for both the object and the subject. The class `rdfs:Resource` is the class of everything in RDF. All things that are described in RDF are instances of this class.

`rdfs:isDefinedBy` is a property that is used to indicate a resource that defines the subject. This is similar to a subject indicator in topic maps, and can therefore be translated into a subject identifier.

The `rdfs:member` property is used to specify members of the subject, and is translated into an association. This association clearly needs role types that specify some sort of directionality, since one needs to know which resource is the member and which resource includes the member. Therefore this association is best expressed using the default association roles.

The last property is `rdfs:subPropertyOf`, which is used to express a hierarchy of properties. Properties are either mapped to occurrence types or association types, which are both actually topics. Because of this, one can use the same subtype-supertype association in topic maps that the property `rdfs:subClassOf` is mapped to.

Table 6 gives a summery of how properties in RDFS can be mapped to topic maps (xtm is the namespace for "http://www.topicmaps.org/xtm/1.0/core.xtm#", which defines PSIs for XTM).

Property	**Mapped to**	**Association type**	**Subject-role in association**	**Object-role in association**
rdfs:subClassOf	Association	xtm:superclass-subclass	xtm:subclass	xtm:superclass
rdfs:label	Base name			
rdfs:comment	Occurrence			
rdfs:domain	Association	rdfs:domain	rdfs:Class	rdfs:Property
rdfs:range	Association	rdfs:range	rdfs:Class	rdfs:Property
rdfs:seeAlso	Association	rdfs:seeAlso	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy	Subject identifier			
rdfs:member	Association	rdfs:member	rtm:subject	rtm:object
rdfs:subPropertyOf	Association	xtm:superclass-subclass	xtm:subclass	xtm:superclass

Table 6: How RDFS properties can be mapped to topic maps

8.3 From OWL to Topic Maps

OWL contains a lot more than topic maps, and finding equivalent topic map constructs to map to is not always possible. Topic maps do not have a constraint language yet, and therefore one can not express the same constraints that one can with OWL. One can simply view OWL as an RDF vocabulary and map properties to occurrences, associations, base names and so on. And classes can be mapped to topics. So the information in OWL can still be mapped to topic maps, but regular topic map applications will not be able to interpret it "correctly".

In OWL you have the possibility to state that two things are to be considered equal by using the property `owl:sameAs`. In topic maps, all things are considered to be different, or represent different subjects. If two topics represent the same subject, they should be merged. So if the `owl:sameAs` property is mapped to a subject identifier in topic maps, it cause the two subject and object of a statement using this property to be merged, and this will fit better with the topic map paradigm.

In OWL, you can define classes using set operators. These set operators are the properties `owl:unionOf`, `owl:intersectionOf`, and `owl:ComplementOf`. In addition, you can create an enumeration class by using the property `owl:oneOf`. Topic maps do not have

anything equivalent to these properties, but they can be mapped to associations with the property is the association type. owl:unionOf specifies that a class is the union of several other classes that are specified using a list in RDF. The domain and range of this property are owl:Class and rdf:List, and these can be used to specify the roles in the association. owl:intersectionOf and owl:oneOf can be mapped in the same way. The domain and range of owl:ComplementOf is both owl:Class. But, if one class is the complement of the other, then the other class is also the complement of the first class, so in this association one does not need to indicate directionality with the association roles, and the role types of both the object and the subject of the statement can be owl:Class.

Cardinality restrictions are expressed in OWL with the owl:Restriction class using the properties owl:Cardinality, owl:minCardinality and owl:maxCardinality, which define the exact, minimum and maximum cardinalities, respectively. Since topic maps do not use cardinality restrictions, and the objects of the statements using these properties are always literals, these properties are best mapped to occurrences.

Other properties in addition to cardinalities that can be used with the owl:Restriction class are owl:onProperty, owl:allValuesFrom, owl:someValuesFrom, and owl:hasValue. The three first properties can simply be mapped to associations in topic maps and use the domain and range as subject and object role types. The last property, however, can not be mapped to an association. This is because it can have either a resource or a literal as the object, and an association in topic maps can not have a literal as a member. So this property must be mapped to an occurrence. This is not a clean mapping, however. The object of this property can be a resource, which would normally translate to a topic in topic maps. A topic can not be an occurrence, but it is possible to have a URI as an occurrence, and this URI could be the identifier of the topic representing the resource. An occurrence that is a URI is supposed to link to an external resource, and in this case it would link to an internal topic, so that is why this translation can not be considered to be clean.

The owl:differentFrom property is used to state that two individuals are distinct. In OWL this is important to use because OWL does not assume that individuals are different. Under certain conditions it may be asserted that two individuals represent the same thing unless it is otherwise explicitly stated. Topic maps do make the assumption that all topics represent different subjects, and therefore this is a property that is not needed in topic maps, and can be ignored in a translation. The property owl:distinctMembers can be used instead of owl:differentFrom if one wants to state that a group of individuals are all different from each other. So one would think that this property also could be ignored when doing a translation. The syntax used with this property makes it a bit more difficult. owl:distinctMembers has domain owl:AllDifferent and range rdf:List. So if one wants to state that the individuals P25 and P26 are different from one another, one could create an instance of the class owl:AllDifferent. This instance would be the subject in a statement with owl:distinctMembers as the predicate, and the object would be an anonymous node that represents the list containing the two individuals P25 and P26. If we simply decided to ignore the property owl:distinctMembers, the instance of owl:AllDifferent and the list of individuals would still be translated, but there would be no connection between them, which would make the translation difficult to understand. Using the RTM vocabulary there is no way one can state that one wants to ignore classes, which would have solved this problem if one could ignore the class owl:AllDifferent. Because of this, owl:distinctMembers is translated into an association, even though the

information it gives is not really needed in topic maps. The domain and range are used for the role types of the subject and the object.

As Garshol suggested, all metadata properties for ontologies are simply mapped to associations with the default role types. Garshol also suggests that one can ignore the property `owl:inverseOf` in topic maps. I am not sure I completely agree. An association in topic maps do not have directionality, but statements in RDF do. For this reason, when a statement in RDF is translated to an association in topic maps, some directionality is still implied when using the default role types of `rtm:subject` and `rtm:object`. When the default role types are used, it could be useful information to state that one association is the "inverse" of another. So I think this property should be mapped to an association with the role type `owl:ObjectProperty` for both the subject and the object.

Table 7 gives a summery of the proposed mappings from OWL properties to topic map constucts.

Property	Mapped to	Association type	Subject-role in association	Object-role in association
owl:equivalentClass	Association	owl:equivalentClass	Owl:Class	owl:Class
owl:disjointWith	Association	owl:disjointWith	Owl:Class	owl:Class
owl:equivalentProperty	Association	owl:equivalentProperty	rdf:Property	rdf:Property
owl:sameAs	Subject identifier			
owl:differentFrom	Ignored			
owl:distinctMembers	Association	owl:distinctMembers	owl:AllDifferent	rdf:List
owl:unionOf	Association	owl:unionOf	owl:Class	rdf:List
owl:intersectionOf	Association	owl:intersectionOf	owl:Class	rdf:List
owl:complementOf	Association	owl:complementOf	owl:Class	owl:Class
owl:oneOf	Association	owl:oneOf	owl:Class	rdf:List
owl:onProperty	Association	owl:onProperty	owl:Restriction	rdf:Property
owl:allValuesFrom	Association	owl:allValuesFrom	owl:Restriction	rdfs:Class
owl:hasValue	Occurrence			
owl:someValuesFrom	Association	owl:someValuesFrom	owl:Restriction	rdfs:Class
owl:minCardinality	Occurrence			
owl:maxCardinality	Occurrence			
owl:cardinality	Occurrence			
owl:inverseOf	Association	owl:inverseOf	owl:ObjectProperty	owl:ObjectProperty
owl:imports	Association	owl:imports	rtm:subject	rtm:object
owl:versionInfo	Occurrence			
owl:priorVersion	Association	owl:priorVersion	rtm:subject	rtm:object
owl:backwardCompatibleWith	Association	owl:backwardCompatibleWith	rtm:subject	rtm:object
owl:incompatibleWith	Association	owl:incompatibleWith	rtm:subject	rtm:object

Table 7: How OWL properties can be mapped to topic maps

8.4 Discussion

The mappings that have been proposed in this chapter are suggested for making the result of a translation from OWL to topic maps fit the topic map paradigm closest possible and to increase the readability of the resulting model. The translation does not give any loss of information since every RDF, RDFS and OWL property are mapped to a topic map construct. Some of the properties in OWL have equivalent constructs in topic maps, while others do not and have to be translated into some other topic characteristic that fit the closest.

By using the domain and range of a property as the association roles when a property is translated into a topic map association, readability of the resulting model can be improved. This is because sometimes the domain and range provide more intuitive roles than `rtm:subject` and `rtm:object`. Also, sometimes a property in OWL can be bidirectional as well, for instance `owl:ComplementOf`, and the directionality implied by using the default association roles may provide more confusion than clarity. However, the suggested mappings are not very well suited if one is interested in reversibility. This is because it is not possible to see what will be the subject and object of a statement, unless this information has been kept somehow when performing the first translation. So the resulting model from a round-trip will most likely not be the same as the original model.

The OWL properties that have equivalent constructs in topic maps should be mapped accordingly to make the result fit better with the topic map paradigm, which will make it easier to use the result of a translation with topic map tools. This may also make reversibility more difficult. For instance, the `owl:sameAs` property is mapped to a subject identifier in topic maps. This results in a topic having more than one identifier, and when translating back to OWL again, one does not know which identifier to use as the URI.

In our case, reversibility of the translation is not that important because we assume that the model will not be changed during use. If reversibility was important, one would have to rethink some of the mapping proposals.

In the post processing phase of a rescue scenario one might want to perform reasoning over the model again with all the new data added. This may help in analyzing how the operation went and what kind of data has been collected. Since the ontology itself will not have changed, one can still use the original ontology expressed in OWL. Since both OWL and topic maps make use of URIs as identifiers, it should not be difficult to add the data, or instances, that has been collected throughout the operation, to the model. And just like in the a priori phase, reasoning can be done on a stronger machine.

When adding data during use in a rescue scenario, I am assuming that it is only allowed to add instances of the defined classes in an ontology, associations between instances and occurrences using the defined properties. The mapping proposals suggested in this chapter are only for the built-in properties of RDF, RDFS, and OWL, not for the properties that may be defined in specific ontologies. Therefore, I am assuming that the ontology specific properties will only use default mappings, i.e. datatype properties will map to occurrences and object properties will map to associations with the default association role types. One could choose to use the domain and range of the properties as the association roles, but in many cases the domain and range may not be specified, and if they were, one would have to specify for the translation to topic maps what association roles are to be the subject and the object of a statement.

The topic types, occurrence types and association types can be specified using the URIs of the corresponding classes and properties in OWL. If the added data is saved separately from the ontology, it can be serialized into XTM, and can be translated to RDF/XML using the Omnigator. This translation is much simpler than having to translate the entire ontology because the association role types specify what to use as the subject and object in an RDF statement. In order to make the translation as simple as possible, I am assuming that topics only have one subject identifier and that scope and variant names are not used. This is

something that would have to be controlled by an application. If one wanted to be able to have multiple subject identifiers, one could just pick one arbitrarily to be the URI when translating to OWL, and use the `owl:sameAs` property for the other identifiers. Base names can be translated into a predefined OWL property, for instance `rdfs:label`, and if n-ary associations occur, they can be translated into multiple statements. In a way, instances are added similarly to how they would be added in OWL. It is a downside that one cannot take advantage of all of the capabilities of topic maps, but that would complicate the translation back to RDF/OWL. This is because RDF and OWL do not have anything equivalent to scope and variant names, and since the ontology is originally created in OWL, these are constructs that are not “expected” in the ontology.

9. Implementation

In the previous chapter we saw that a translation from topic maps to OWL is possible and can be done without loss of information, but the question remains of how usable the result of such a translation is. In this chapter I will use the suggested mappings from chapter 8 to translate the OWL ontologies from chapter 5 to topic maps. I will use Ontopia's Omnigator to perform the translation, and give it the mappings expressed in RDF/XML that can be found in Appendix B, which is the same information that was given in chapter 8. The Omnigator can take a file in the RDF/XML format, translate it into a topic map, and export it to the XTM format.

Ontology name	OWL		Topic maps			
	RDF Triples	RDF/XML File size (KB)	Topics	Occurrences	Associations	XTM file size (KB)
Body Sensor	67	6	46	3	35	32
Alertness Status	51	3	41	0	32	26
Sensor Upper	31	3	30	1	16	18
Epicrisis	13	2	23	1	7	12
Person	10	2	18	0	4	9
Rescue	24	3	32	0	16	19

Table 8: Statistics for translating from OWL to topic maps

Table 8 shows some statistics from the translation of the ontologies. The most obvious difference is the file size which is much larger for the resulting topic map than the original OWL ontology. This is most likely because of the fact that in OWL you have a lot of built-in classes and properties, which all have to be modelled in topic maps when performing a translation. In a MANET, the file size can be of great importance because of memory limitations on devices. Vigdal [2] has done some performance tests on how large topic maps mobile devices are able to handle. He has performed the tests on two different mobile phones, Sony Ericsson T610 and K600i. The T610 model encountered problems with topic maps that consisted of 112 TAOs (topics, associations and occurrences) and that had a file size of 29KB, and would most likely not be able to handle the result of a translation of the Body Sensor ontology. The other phone did not have problems until it was tested with a topic map consisting of 524 TAOs that had a file size of 221KB. The original OWL ontologies used in this translation are all rather small, and the ontologies that will be actually be used in a rescue scenario are most likely larger. So the fact that a translation from OWL to topic maps creates much larger files needs to be taken under consideration.

We know that all the information that was present in the OWL ontology, will also be present in the topic map that results from a translation. An important issue is how usable this information is. Figure 25 shows the Person ontology after it has been translated to topic maps. The figure is taken from Ontopia's Vizigator. The ellipses represent topics and the lines between them represent associations. The names in the ellipses are base names that are

generated by the Omnigator which are based on the URIs of the resources. It is not possible in this figure to see the association types or association roles, but this is possible when using the Vizigator by moving the mouse over an association.

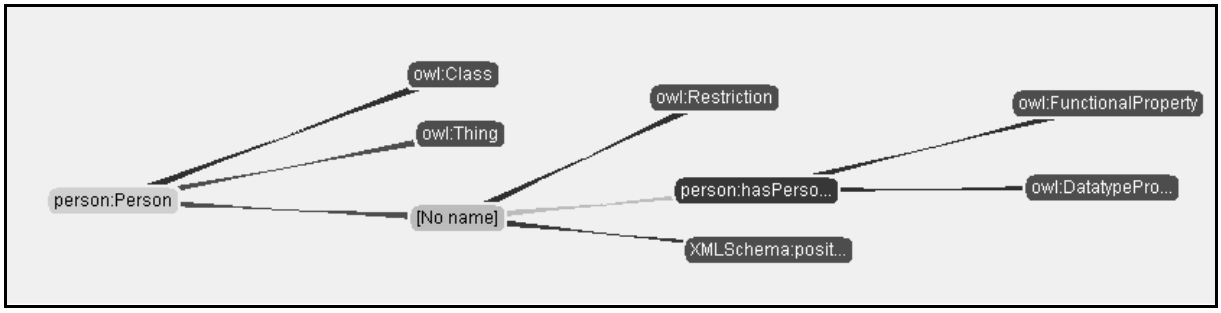

Figure 25: Person ontology in topic maps

So what information can we derive from Figure 25. Firstly, there is a striking resemblance between this topic map and the way that the original ontology can be represented in an RDF graph, see Figure 14. Since OWL is expressed using RDF/XML, it can be parsed into RDF triples and represented as an RDF graph. The Semantics and Abstract Syntax for OWL [61] say that one must take great care when representing OWL as an RDF graph. This is because all of the built-in classes and properties of OWL are modelled in the graph, and it makes it possible and easy to make new statements about them when editing the ontology. These classes and properties should not under any circumstances be changed because that would change OWL itself. The same can be applied when OWL classes and properties are modelled in topic maps, and is another good reason for why the model should not be allowed to be changed during use in a rescue scenario.

From the resemblance between the topic map model and the RDF graph, one could conclude that the resulting topic map from a translation corresponds well with the RDF/OWL model. So the question is how well it corresponds with the topic map paradigm. Regular topic map constructs are used, and equivalences between the two paradigms are used when possible, so there is no reason why the result should not correspond well with the topic map paradigm. The only problem is that of interpretation of the information in the model.

Looking at Figure 25, we can see that there is a topic called `person:Person` that is an instance of `owl:Class`, and a subclass of `owl:Thing`. `person:Person` is also a subclass of a topic that does not have a name and that is an instance of `owl:Restriction`. A person that is familiar with OWL could derive from this information that there is an OWL class called `person:Person`, and this class has a restriction on a property, or association in topic maps. However, the typical users in a rescue scenario are rescue personnel, and they can not be expected to know much about either topic maps or OWL. Such a user would not be able to make much sense of the information in the topic map model.

When using the topic maps paradigm, it is recommended that users are not in direct contact with the topic map model, but an application that uses the model. The topic map application can make it easier for users to browse the topic map in a way that makes sense to them and gives them knowledge without having to have much, if any, knowledge about topic maps. The same applies to the usage of OWL. But, when translating from topic maps to OWL, OWL concepts are mixed in with the topic map. One can not expect an application created for topic maps to be able to understand OWL concepts, and vice versa. For instance, in OWL you can create restrictions on properties. Applications for OWL are designed to be able to make sure

that restrictions are upheld when using an ontology. When translating the same information to topic maps, a topic map application does not know what the restrictions mean and will therefore not be able to make sure that the restriction is upheld. So then it would be up to the user to recognize the restrictions and make sure that they are upheld. Leaving such an amount of responsibility on the user can cause the user to be frustrated because there is too much information to consider. The only way to solve this problem would be to have a specialized application for topic maps that understood the meaning of OWL properties and classes.

Such an application could query the model for any restrictions whenever new data was being added. In practice this would mean querying for any supertypes that were instances of `owl:Restriction`, checking what properties the restrictions were associated with and what possible values they may have. Then the user would only be allowed to add data that was consistent with the restrictions. For instance, say that a user wants to add an instance of `pers:Person`. The application would then search for restrictions, and find that there was a restriction with regards to the property `pers:hasPersonId`. Further it could find the information that this property is a functional property. The user would then have to be informed that he or she could only add one social security number and that it has to be a positive integer, and the application would have to make sure that the restrictions were actually upheld.

The method that I have just described adds a lot of overhead in that multiple queries will have to be made, which may be time consuming and therefore resource constraining. A different approach is if one knows all of the ontologies that will be used beforehand, and then create the application interface in such a way that only legal values were allowed to be added. This does not require querying of the model every time data is to be added, but it has the disadvantage that one needs to know all of the ontologies that will be used beforehand, and if one wants to change the ontologies some time, a lot of change must be done to the application as well.

Neither of the approaches above can assure complete consistency of the model. This is because different devices may have different information about the same or different resources. The information on one device may be consistent with the ontology, but when it is combined with the information from other devices, it can become inconsistent. Imagine that a property is defined as being functional in OWL. On two different devices, they can, for the same subject, have given a different value for this property. Then, if this information is combined, you have the case that a functional property has two different values, which is inconsistent with the definition of the property. A similar problem may occur if two classes are defined to be disjoint, and the same instance is defined to be a member of both classes on different devices. On a single device one can make sure that an instance is not a member of two disjoint classes, but one can not be sure that this is upheld across different devices.

The problems mentioned above are caused by the fact that data is distributed across different devices. This is not a problem if one is using regular topic maps, since they do not have restrictions, but become problems when using OWL or OWL translated to topic maps with a desire to keep the restrictions. One can either choose to accept the fact that inconsistencies may occur, or one can choose not to create the restrictions that may cause these inconsistencies, that would mean to not have any kind of cardinality restrictions, disjointness, or complementary classes. None of these proposals are really satisfying, since in one case you must accept that there may be inconsistencies and in the other case you lose a lot of expressivity. A different approach could be to add timestamps to all information stating when

the information was added. This way, if an inconsistency was discovered, for instance that a functional property has two different values, one could check the timestamp for when the information was added and use the information that was added most recent. One would then have to assume that the information that was added most recent would also be the most correct. This approach is not ideal either, because timestamps have to be saved for every time information is added, which requires extra memory space. But at least one can make sure that the ontology remains consistent when all of the information from all of the devices is joined together.

10. Conclusion and further work

10.1 Conclusion

OWL can be used as an ontology language to describe different concepts in a rescue scenario, and provide translations between different vocabularies used in different organizations. OWL is built on the languages RDF and RDFS, and is constructed as an RDF vocabulary. With OWL one can create restrictions for classes, and create complex classes because it is a very expressive language. Reasoning can be performed to check for consistency and infer new knowledge.

Topic maps can also be used as an ontology language, but is less expressive than OWL and does not have support for reasoning. The advantage of topic maps is that there exists a topic map engine that can be used on resource-limited devices.

In this thesis we have looked at how one can translate OWL ontologies to topic maps for use in a rescue scenario. This approach was taken because a tool for OWL that could work on a resource-limited device was not discovered until late in the work. Some reasoning tasks for OWL ontologies can be done before an ontology is translated into a topic map, like concept subsumption and concept satisfiability testing, assuming that the ontology will not change during a rescue operation. Therefore, a translation from OWL to topic maps can contain the knowledge inferred from the reasoning engine, and can then be browsed and queried using a topic map engine on resource-limited devices.

There have been proposals for translation between RDF and topic maps, but these were written before the standards were formalized, and are therefore not complete translations. The proposals were either suggesting a semantic mapping, finding equivalent constructs in both languages, or an object mapping, where one of the languages is modelled in the other language. A semantic mapping seems to be the better choice because it gives a much more natural result. Only one proposal has made suggestions for how OWL can be translated into topic maps, but it only suggests translations for some of the properties in OWL, not all of them. A complete semantic mapping from OWL to topic maps is not possible because OWL is a lot more expressive and there are not equivalent constructs in topic maps for all the constructs in OWL.

We have made suggestions for how the different properties that are built into RDF, RDFS and OWL can be mapped to topic maps, using equivalent concepts where possible, and where this is not possible, translating into occurrences or associations with the URI of the OWL property becoming the occurrence or association type. In some cases, one can use the domain and range of a property to specify the association role types if a property in OWL is translated into an association in topic maps. This can increase the readability of the model.

Results of actual translations show the file size increases when translating from OWL to topic maps. This is because all of the concepts that are built into OWL have to be modelled in the resulting topic map. File size is of big importance when using resource-limited devices because they are limited in memory and therefore not capable of handling large files.

Even though a translation from OWL to topic maps is possible without loss of information, the interpretation of the resulting topic map can not be done by generic topic map applications. This is because topic maps do not have restrictions and many of the other

properties in OWL, and in order to make sure that restrictions are upheld and that inconsistencies are not created, one needs applications for topic maps that understands the OWL model and can be responsible for upholding restrictions. So it is possible to perform a translation, but it is not an ideal solution since it requires specialized applications to interpret the resulting model.

10.2 Further work

As the situation is now, a translation from OWL to topic maps is probably not the best solution for how one can use OWL on resource-limited devices. OWL is just too expressive for a good translation to be made. A constraint language for topic maps is currently being worked on, the Topic Map Constraint Language, or TMCL. When this language is formalized, there may be a better translation from OWL to topic maps, because TMCL and OWL may have more equivalent concepts.

This thesis was written based on the assumption that there does not exist any tools for OWL that can be used on resource-limited devices. This assumption was made after having done a lot of search for such tools or research on how it could be done. It was only discovered later that a description logic reasoner, Pocket KRHyper, had been created that was explicitly designed to work on resource-limited devices. This reasoner does not have direct support for OWL, but it would be interesting to look at implementing either an interface for OWL or a DIG interface and test it with an application using OWL ontologies.

There is also a lot of research that can be done in order to find out more about the type of information that is most important in a rescue scenario where the personnel are communicating by using handheld devices. This involves looking into what types of queries will most likely be used, which is important in order to predict what kind of reasoning support is needed. If only simple queries are expected, one may only need very simple reasoning support, or if very complex queries are expected, reasoning may be more difficult. Currently, rescue personnel are using only oral communication, so there is very little knowledge about what kind of information they would expect to be able to retrieve when using handheld devices.

11. References

1. Andersen, L.J.B.: *Merging Topic Maps on Mobile Devices.* Master thesis. University of Oslo. 2006

2. Vigdal, S.: *Informasjonsdeling i redningsarbeid ved bruk av Emnekart (Topic Maps).* Master thesis. University of Oslo. 2006

3. Heflin, J.: *OWL Web Ontology Language - Use Cases and Requirements*. [online] 10.02.2004. Available at: http://www.w3.org/TR/webont-req/ [Last accessed: 26.04.2007]

4. Montanari, R., Toninelli, A., Corradi, A.: "Adaptive semantic support provisioning in mobile Internet environments". In *Proceedings of The 2005 Symposium on Applications and the Internet Workshops***.** 2005.

5. Chen, H., Finin, T., Joshi, A.: *The Role of the Semantic Web in Pervasive Context-Aware Systems*. [online] 2003. Available at: http://www.csee.umbc.edu/~finin//papers/iswc03CobraDraft.pdf [Last accessed: 26.04.2007]

6. Grau, B.C.: *OWL 1.1 Web Ontology Language Tractable Fragments*. [online] 06.04.2007. Available at: http://webont.org/owl/1.1/tractable.html [Last accessed: 24.04.2007]

7. *EXPTIME - From Wikipedia, the free encyclopedia*. [online] Available at: http://en.wikipedia.org/wiki/EXPTIME [Last accessed: 25.04.2007]

8. *NEXPTIME - From Wikipedia, the free encyclopedia*. [online] Available at: http://en.wikipedia.org/wiki/NEXPTIME [Last accessed: 25.04.2007]

9. Gruber, T.: *What is an Ontology?* [online] Available at: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html [Last accessed: 16.04.2007]

10. Gruber, T.: *It Is What It Does: The Pragmatics of Ontology as Language, Contract, and Content*. [online] 24.08.2000. [Slides from presentation] Available at: http://www.cs.man.ac.uk/~stevensr/workshop/gruber.zip [Last accessed: 17.04.2007]

11. Ontology (computer science) - From Wikipedia, the free encyclopedia. [online] Available at: http://en.wikipedia.org/wiki/Ontology_(computer_science) [Last accessed: 21.04.2007]

12. Sanderson, N., Goebel, V., Munthe-Kaas, E.: "Metadata Management for Ad-Hoc InfoWare - A Rescue and Emergency Use Case for Mobile Ad-Hoc Scenarios". In *International Conference on Ontologies, Databases and Applications of Semantics (ODBASE05)***.** Cyprus, 2005.

13. Munthe-Kaas, E., Drugan, O., Goebel, V., Plagemann, T., Pužar, M., Sanderson, N., Skjelsvik, K.S.: "Mobile Middleware for Rescue and Emergency

Scenarios". In: Bellavista, P., Corradi, A. (eds.): *Mobile Middleware*. CRC Press (2006)

14. *Semantic Web Activity Statement*. [online] Available at: http://www.w3.org/2001/sw/Activity [Last accessed: 26.04.2007]

15. Berners-Lee, T., Hendler, J., Lassila, O.: "The Semantic Web". In *Scientific American.* Issue May 2001.

16. Manola, F., Miller, E.: *RDF Primer*. [online] 10.02.2004. Available at: http://www.w3.org/TR/rdf-primer/ [Last accessed: 26.04.2007]

17. *RDF Vocabulary Description Language 1.0: RDF Schema*. [online] 10.02.2004. Available at: http://www.w3.org/TR/rdf-schema/ [Last accessed: 19.04.2007]

18. Smith, M.K., Welty, C., McGuinness, D.L.: *OWL Web Ontology Language Guide*. [online] 10.02.2004. Available at: http://www.w3.org/TR/owl-guide/ [Last accessed: 26.04.2007]

19. McGuinness, D.L., Harmelen, F.v.: *OWL Web Ontology Language Overview*. [online] 10.02.2004. Available at: http://www.w3.org/TR/owl-features/ [Last accessed: 26.04.2007]

20. Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C.: *A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools*. [online] Available at: http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf [Last accessed: 19.04.2007]

21. Tobies, S.: *Complexity Results and Practical Algorithms for Logics in Knowledge Representation.* PhD thesis. RWTH Aachen University, Germany. 2001

22. Bechhofer, S.: *The DIG Description Logic Interface: DIG/1.0*. [online] 01.10.2002. Available at: http://dl-web.man.ac.uk/dig/2002/10/interface.pdf [Last accessed: 26.04.2007]

23. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: *Pellet: A Practical OWL-DL Reasoner*. [online] Available at: http://www.mindswap.org/papers/PelletJWS.pdf [Last accessed: 23.04.2007]

24. RacerPro. [online] [Website for the RacerPro reasoner] Available at: http://www.racer-systems.com/ [Last accessed: 19.04.2007]

25. FaCT++. [online] [Website for the Fact++ reasoner] Available at: http://owl.man.ac.uk/factplusplus/ [Last accessed: 24.04.2007]

26. KAON2. [online] [Website for the KAON2 reasoner] Available at: http://kaon2.semanticweb.org/ [Last accessed: 12.04.2007]

27. Jena – A Semantic Web Framework for Java. [online] [Project website] Available at: http://jena.sourceforge.net/ [Last accessed: 21.01.2007]

28. Pepper, S., Moore, G.: *XML Topic Maps (XTM) 1.0*. [online] 2001.
Available at: http://www.topicmaps.org/xtm/1.0/ [Last accessed: 23.03.2007]

29. Pepper, S.: *The TAO of Topic Maps*. [online]
Available at: http://www.ontopia.net/topicmaps/materials/tao.html
[Last accessed: 03.03.2007]

30. *Guide to the topic map standards*. [online] 23.06.2002.
Available at: http://www1.y12.doe.gov/capabilities/sgml/sc34/document/0323.htm
[Last accessed: 26.04.2007]

31. International Organization for Standardization. [online] [Organization website]
Available at: http://www.iso.org/ [Last accessed: 26.04.2007]

32. *ISO/IEC 13250 Topic Maps*. [online] 03.12.1999.
Available at: http://www1.y12.doe.gov/capabilities/sgml/sc34/document/0129.pdf
[Last accessed: 26.04.2007]

33. *HyTime - From Wikipedia, the free encyclopedia*. [online]
Available at: http://en.wikipedia.org/wiki/HyTime [Last accessed: 26.04.2007]

34. *Standard Generalized Markup Language - From Wikipedia, the free encyclopedia*.
[online] Available at: http://en.wikipedia.org/wiki/SGML [Last accessed: 01.04.2007]

35. Pepper, S., Naito, M.: *Topic Maps - Overview and Basic Concepts*. [online]
03.11.2003.
Available at: http://www1.y12.doe.gov/capabilities/sgml/sc34/document/0446.htm
[Last accessed: 11.03.2007]

36. Garshol, L.M., Moore, G.: *ISO 13250-2: Topic Maps — Data Model*. [online]
18.06.2006. Available at: http://www.isotopicmaps.org/sam/sam-model/
[Last accessed: 03.04.2007]

37. Garshol, L.M., Moore, G.: *Topic Maps - XML Syntax*. [online] 19.06.2006.
Available at: http://www.isotopicmaps.org/sam/sam-xtm/ [Last accessed: 26.04.2007]

38. *Topic Maps — Canonicalization*. [online] 01.11.2004.
Available at: http://www.isotopicmaps.org/sam/cxtm/ [Last accessed: 03.03.2007]

39. *Topic Maps Reference Model, 13250-5*. [online] 11.08.2006.
Available at: http://www.isotopicmaps.org/TMRM/TMRM-latest.pdf
[Last accessed: 26.04.2007]

40. Garshol, L.M.: *Why XTM 2.0 is different from 1.0*. [online] 16.12.2006
Available at: http://www.garshol.priv.no/blog/85.html [Last accessed: 04.04.2007]

41. Schwotzer, T., Geihs, K.: "Shark – a System for Management, Synchronization and Exchange of Knowledge in Mobile User Groups". In *Journal of Universal Computer Science.* Vol. 8, issue 6 (2002) p. 644-651.

42. Sinner, A., Kleemann, T.: "KRHyper - In Your Pocket, System Description". In *Proc. of Conference on Automated Deduction, CADE-20***.** Tallinn, Estonia, 2005.

43. Kleemann, T., Sinner, A.: "Description logic based matchmaking on mobile devices." In *Proc. of 1st Workshop on Knowledge Engineering and Software Engineering, KESE2005*. Koblenz, Germany, 2005.

44. Kleemann, T., Sinner, A.: *User Profiles and Matchmaking on Mobile Phones*. [online] 19.09.2005.
Available at: http://www.uni-koblenz.de/~tomkl/Publ/KleemannSinnerINAP05.pdf
[Last accessed: 21.03.2007]

45. Horrocks, I.: *Hybrid Logics and Ontology Languages*. [online] 12.08.2006.
[Slides from presentation]
Available at: www.cs.man.ac.uk/~horrocks/Slides/HyLo06.ppt
[Last accessed: 07.03.2007]

46. Protégé. [online] [Website of project]
Available at: http://protege.stanford.edu/ [Last accessed: 21.04.2007]

47. Garshol, L.M.: *Topic Maps – Ontologies for Humans?* [online] 08.12.2005.
[Slides from presentation]
Available at:
http://www.xmluk.org/images/content/Ontologies_and_XML_12_05/tms-ontologies-for-humans.pdf [Last accessed: 18.03.2007]

48. Garshol, L.M.: Topic maps, RDF, DAML, OIL - A comparison. [online]
Available at: http://www.ontopia.net/topicmaps/materials/tmrdfoildaml.html
[Last accessed: 26.03.2007]

49. Garshol, L.M.: *Living with topic maps and RDF*. [online]
Available at: http://www.ontopia.net/topicmaps/materials/tmrdf.html
[Last accessed: 21.03.2007]

50. Pepper, S.: SWBPD: RDF/Topic Maps Interoperability Task Force. [online]
Available at: http://www.w3.org/2001/sw/BestPractices/RDFTM/
[Last accessed: 26.04.2007]

51. Pepper, S., Vitali, F., Garshol, L.M., Gessa, N., Presutti, V.: *A Survey of RDF/Topic Maps Interoperability Proposals* [online] 10.02.2006.
Available at: http://www.w3.org/TR/2006/NOTE-rdftm-survey-20060210/
[Last accessed: 21.03.2007]

52. *Guidelines for RDF/Topic Maps Interoperability*. [online] 30.06.2006.
Available at: http://www.w3.org/2001/sw/BestPractices/RDFTM/guidelines-20060630.html [Last accessed: 03.03.2007]

53. Moore, G.: *RDF and TopicMaps - An Exercise in Convergence*. [online] 2001.
Available at: http://xml.coverpages.org/moore-topicmapsrdf200105.pdf
[Last accessed: 05.04.2007]

54. Ontopia: *RTM: An RDF-to-TM mapping*. [online] 17.11.2003.
Available at: http://psi.ontopia.net/rdf2tm/ [Last accessed: 02.04.2007]

55. *The Ontopia Schema Language - Tutorial*. [online] 08.11.2006.
Available at: http://www.ontopia.net/omnigator/docs/schema/tutorial.html
[Last accessed: 26.04.2007]

56. Ontopia: TMR: A TM-to-RDF mapping. [online]
Available at: http://psi.ontopia.net/tm2rdf/ [Last accessed: 26.04.2007]

57. tmapi-utils. [online] [Project website]
Available at: http://sourceforge.net/projects/tmapi-utils/ [Last accessed: 02.02.2007]

58. TMAPI - Common Topic Map Application Programming Interface. [online]
[Project website] Available at: http://tmapi.org/ [Last accessed: 26.03.2007]

59. tinyTIM - a tiny Topic Map Engine & TMAPI implementation. [online]
[Project website]
Available at: http://tinytim.sourceforge.net/ [Last accessed: 26.03.2007]

60. TM4J - Topic Maps For Java. [online] [Project website]
Available at: http://tm4j.org/ [Last accessed: 26.03.2007]

61. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: *OWL Web Ontology Language - Semantics and Abstract Syntax* [online] 10.02.2004
Available at: http://www.w3.org/TR/owl-semantics/ [Last accessed: 26.04.2007]

Appendix A Ontologies expressed in RDF/XML

This appendix contains examples of OWL ontologies that may be used in a rescue scenario. They are expressed in RDF/XML, and described in chapter 5.

Person ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://www.ifi.uio.no/dmms/epicrisis_ex/person#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xml:base="http://www.ifi.uio.no/dmms/epicrisis_ex/person">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Person">
    <rdfs:subClassOf
           rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:ID="hasPersonId"/>
        </owl:onProperty>
        <owl:someValuesFrom
   rdf:resource="http://www.w3.org/2001/XMLSchema#positiveInteger"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:FunctionalProperty rdf:about="#hasPersonId">
    <rdf:type
     rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
</rdf:RDF>
```

Epicrisis ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:pers="http://www.ifi.uio.no/dmms/epicrisis_ex/person#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.ifi.uio.no/dmms/epicrisis_ex/epicrisis#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xml:base="http://www.ifi.uio.no/dmms/epicrisis_ex/epicrisis">
  <owl:Ontology rdf:about="">
    <owl:imports
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/person"/>
  </owl:Ontology>
  <owl:Class rdf:ID="Epicrisis">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/person#Person"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="ofTargetPerson"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#ofTargetPerson"/>
        <owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
</rdf:RDF>
```

Alertness status ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:pers="http://www.ifi.uio.no/dmms/epicrisis_ex/person#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.ifi.uio.no/dmms/epicrisis_ex/epicrisis#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xml:base="http://www.ifi.uio.no/dmms/epicrisis_ex/epicrisis">
  <owl:Ontology rdf:about="">
    <owl:imports
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/person"/>
  </owl:Ontology>
  <owl:Class rdf:ID="Epicrisis">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/person#Person"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="ofTargetPerson"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#ofTargetPerson"/>
        <owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
</rdf:RDF>
```

Sensor upper ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns="http://ifi.uio.no/dmms/epicrisis_ex/sensor#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://ifi.uio.no/dmms/epicrisis_ex/sensor">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Observation">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="atTime"/>
        </owl:onProperty>
        <owl:allValuesFrom
rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="withUnit"/>
        </owl:onProperty>
        <owl:allValuesFrom>
          <owl:Class rdf:ID="UnitOfMeasurement"/>
        </owl:allValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Sensor">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="#Observation"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasObservation"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="hasSensorId"/>
        </owl:onProperty>
        <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty rdf:resource="#hasSensorId"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:DatatypeProperty rdf:ID="withValue"/>
</rdf:RDF>
```

Body Sensor ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:pers="http://www.ifi.uio.no/dmms/epicrisis_ex/person#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:sns="http://ifi.uio.no/dmms/epicrisis_ex/sensor#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns="http://www.ifi.uio.no/dmms/epicrisis_ex/bodySensor#"
  xml:base="http://www.ifi.uio.no/dmms/epicrisis_ex/bodySensor">
  <owl:Ontology rdf:about="">
    <owl:imports
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/sensor"/>
  </owl:Ontology>
  <owl:Class rdf:ID="HeartRateObservation">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#withUnit"/>
        <owl:hasValue>
          <sns:UnitOfMeasurement rdf:ID="bpm">
            <owl:sameAs>
              <sns:UnitOfMeasurement rdf:ID="beatsPerMinute">
                <owl:sameAs rdf:resource="#bpm"/>
              </sns:UnitOfMeasurement>
            </owl:sameAs>
          </sns:UnitOfMeasurement>
        </owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom
rdf:resource="http://www.w3.org/2001/XMLSchema#positiveInteger"/>
        <owl:onProperty
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#withValue"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#Observation"/>
  </owl:Class>
  <owl:Class rdf:ID="PulseOxiometryObservation">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom
rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
        <owl:onProperty
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#withValue"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#withUnit"/>
        <owl:hasValue>
          <sns:UnitOfMeasurement rdf:ID="percentOfNormal"/>
```

```
        </owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#Observation"/>
  </owl:Class>
  <owl:Class rdf:ID="BodySensor">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasOxygenSaturation"/>
        </owl:onProperty>
        <owl:allValuesFrom rdf:resource="#PulseOxiometryObservation"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#Sensor"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasHeartRate"/>
        </owl:onProperty>
        <owl:allValuesFrom rdf:resource="#HeartRateObservation"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasBodyTemperature"/>
        </owl:onProperty>
        <owl:allValuesFrom>
          <owl:Class rdf:ID="BodyTemperatureObservation"/>
        </owl:allValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#BodyTemperatureObservation">
    <rdfs:subClassOf
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#Observation"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#withUnit"/>
        <owl:hasValue>
          <sns:UnitOfMeasurement rdf:ID="degCelsius"/>
        </owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom
rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
        <owl:onProperty
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#withValue"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="hasPulseOximetry">
    <rdfs:subPropertyOf
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#hasObservation"/>
    <owl:equivalentProperty>
```

```
      <owl:ObjectProperty rdf:about="#hasOxygenSaturation"/>
    </owl:equivalentProperty>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasHeartRate">
    <rdfs:subPropertyOf
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#hasObservation"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasOxygenSaturation">
    <rdfs:subPropertyOf
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#hasObservation"/>
    <owl:equivalentProperty rdf:resource="#hasPulseOximetry"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasBodyTemperature">
    <rdfs:subPropertyOf
rdf:resource="http://ifi.uio.no/dmms/epicrisis_ex/sensor#hasObservation"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Rescue ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:pers="http://www.ifi.uio.no/dmms/epicrisis_ex/person#"
    xmlns:sns="http://www.ifi.uio.no/dmms/epicrisis_ex/sensor#"
    xmlns:epi="http://www.ifi.uio.no/dmms/epicrisis_ex/epicrisis#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cas="http://www.ifi.uio.no/dmms/epicrisis_ex/casualty#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.ifi.uio.no/dmms/epicrisis_ex/rescue#"
  xml:base="http://www.ifi.uio.no/dmms/epicrisis_ex/rescue">
  <owl:Ontology rdf:about="">
    <owl:imports
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/epicrisis"/>
    <owl:imports
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/casualty"/>
    <owl:imports
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/person"/>
    <owl:imports
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/sensor"/>
  </owl:Ontology>
  <owl:Class rdf:ID="Person">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="withSensor"/>
        </owl:onProperty>
        <owl:allValuesFrom
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/sensor#Sensor"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="withEpicrisis"/>
        </owl:onProperty>
        <owl:allValuesFrom
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/epicrisis#Epicrisis"/
>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom
rdf:resource="http://www.ifi.uio.no/dmms/epicrisis_ex/casualty#CasualtyStat
us"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="withCasualtyStatus"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <owl:equivalentClass>
      <rdf:Description
rdf:about="http://www.ifi.uio.no/dmms/epicrisis_ex/person#Person">
        <owl:equivalentClass rdf:resource="#Person"/>
      </rdf:Description>
    </owl:equivalentClass>
```

```
  </owl:Class>
</rdf:RDF>
```

Appendix B Mappings

This appendix contains the mappings expressed in RDF/XML that are used for translating from topic maps to OWL. The mappings are explained in chapter 8.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!DOCTYPE rdf:RDF [
     <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
     <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
     <!ENTITY rtm  "http://psi.ontopia.net/rdf2tm/#" >
     <!ENTITY xtm  "http://www.topicmaps.org/xtm/1.0/core.xtm#" >
   ]>
<rdf:RDF xmlns:rdf="&rdf;"
         xmlns:rtm="&rtm;">

<!-- RDF mappings -->

  <rdf:Description rdf:about="&rdf;type">
    <rtm:maps-to rdf:resource="&rtm;instance-of"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdf;subject">
   <rtm:maps-to rdf:resource="&rtm;association"/>
   <rtm:subject-role rdf:resource="&rdf;Statement"/>
   <rtm:object-role rdf:resource="&rdfs;Resource"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdf;object">
   <rtm:maps-to rdf:resource="&rtm;association"/>
   <rtm:subject-role rdf:resource="&rdf;Statement"/>
   <rtm:object-role rdf:resource="&rdfs;Resource"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdf;predicate">
   <rtm:maps-to rdf:resource="&rtm;association"/>
   <rtm:subject-role rdf:resource="&rdf;Statement"/>
   <rtm:object-role rdf:resource="&rdfs;Resource"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdf;value">
   <rtm:maps-to rdf:resource="&rtm;occurrence"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdf;first">
   <rtm:maps-to rdf:resource="&rtm;association"/>
   <rtm:subject-role rdf:resource="&rdf;List"/>
   <rtm:object-role rdf:resource="&rdfs;Resource"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdf;rest">
   <rtm:maps-to rdf:resource="&rtm;association"/>
   <rtm:subject-role rdf:resource="&rtm;subject"/>
   <rtm:object-role rdf:resource="&rtm;object"/>
  </rdf:Description>
```

```
<!-- RDFS mappings -->

  <rdf:Description rdf:about="&rdfs;label">
    <rtm:maps-to rdf:resource="&rtm;basename"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdfs;comment">
    <rtm:maps-to rdf:resource="&rtm;occurrence"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdfs;subClassOf">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:type rdf:resource="&xtm;superclass-subclass"/>
    <rtm:subject-role rdf:resource="&xtm;subclass"/>
    <rtm:object-role rdf:resource="&xtm;superclass"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdfs;subPropertyOf">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:type rdf:resource="&xtm;superclass-subclass"/>
    <rtm:subject-role rdf:resource="&xtm;subclass"/>
    <rtm:object-role rdf:resource="&xtm;superclass"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdfs;domain">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role rdf:resource="&rdfs;Class"/>
    <rtm:object-role rdf:resource="&rdf;Property"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdfs;range">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role rdf:resource="&rdfs;Class"/>
    <rtm:object-role rdf:resource="&rdf;Property"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdfs;seeAlso">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role rdf:resource="&rdfs;Resource"/>
    <rtm:object-role rdf:resource="&rdfs;Resource"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdfs;isDefinedBy">
    <rtm:maps-to rdf:resource="&rtm;subject-identifier"/>
  </rdf:Description>

  <rdf:Description rdf:about="&rdfs;member">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role rdf:resource="&rtm;subject"/>
    <rtm:object-role rdf:resource="&rtm;object"/>
  </rdf:Description>

<!-- OWL mappings -->

  <rdf:Description rdf:about="&owl;equivalentClass">
```

```
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role  rdf:resource="&owl;Class"/>
    <rtm:object-role  rdf:resource="&owl;Class"/>
  </rdf:Description>

  <rdf:Description rdf:about="&owl;disjointWith">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role  rdf:resource="&owl;Class"/>
    <rtm:object-role  rdf:resource="&owl;Class"/>
  </rdf:Description>

  <rdf:Description rdf:about="&owl;equivalentProperty">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role  rdf:resource="&rdf;Property"/>
    <rtm:object-role  rdf:resource="&rdf;Property"/>
  </rdf:Description>

  <rdf:Description rdf:about="&owl;sameAs">
    <rtm:maps-to rdf:resource="&rtm;subject-identifier"/>
  </rdf:Description>

  <rdf:Description rdf:about="&owl;distinctMembers">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role rdf:resource="&owl;AllDifferent"/>
    <rtm:object-role rdf:resource="&rdf;List"/>
  </rdf:Description>

  <rdf:Description rdf:about="&owl;unionOf">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role rdf:resource="&owl;Class"/>
    <rtm:object-role rdf:resource="&rdf;List"/>
  </rdf:Description>

  <rdf:Description rdf:about="&owl;intersectionOf">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role rdf:resource="&owl;Class"/>
    <rtm:object-role rdf:resource="&rdf;List"/>
  </rdf:Description>

  <rdf:Description rdf:about="&owl;complementOf">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role rdf:resource="&owl;Class"/>
    <rtm:object-role rdf:resource="&owl;Class"/>
  </rdf:Description>

  <rdf:Description rdf:about="&owl;oneOf">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role rdf:resource="&owl;Class"/>
    <rtm:object-role rdf:resource="&rdf;List"/>
  </rdf:Description>

 <rdf:Description rdf:about="&owl;onProperty">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role rdf:resource="&owl;Restriction"/>
    <rtm:object-role rdf:resource="&rdf;Property"/>
  </rdf:Description>
```

```
<rdf:Description rdf:about="&owl;allValuesFrom">
   <rtm:maps-to rdf:resource="&rtm;association"/>
   <rtm:subject-role rdf:resource="&owl;Restriction"/>
   <rtm:object-role rdf:resource="&owl;Class"/>
 </rdf:Description>

 <rdf:Description rdf:about="&owl;hasValue">
   <rtm:maps-to rdf:resource="&rtm;occurrence"/>
 </rdf:Description>

<rdf:Description rdf:about="&owl;someValuesFrom">
   <rtm:maps-to rdf:resource="&rtm;association"/>
   <rtm:subject-role rdf:resource="&owl;Restriction"/>
   <rtm:object-role rdf:resource="&owl;Class"/>
 </rdf:Description>

 <rdf:Description rdf:about="&owl;minCardinality">
   <rtm:maps-to rdf:resource="&rtm;occurrence"/>
 </rdf:Description>

 <rdf:Description rdf:about="&owl;maxCardinality">
   <rtm:maps-to rdf:resource="&rtm;occurrence"/>
 </rdf:Description>

 <rdf:Description rdf:about="&owl;cardinality">
   <rtm:maps-to rdf:resource="&rtm;occurrence"/>
 </rdf:Description>

 <rdf:Description rdf:about="&owl;inverseOf">
   <rtm:maps-to rdf:resource="&rtm;association"/>
   <rtm:subject-role rdf:resource="&owl;ObjectProperty"/>
   <rtm:object-role rdf:resource="&owl;ObjectProperty"/>
 </rdf:Description>

 <rdf:Description rdf:about="&owl;imports">
   <rtm:maps-to rdf:resource="&rtm;association"/>
   <rtm:subject-role rdf:resource="&rtm;subject"/>
   <rtm:object-role rdf:resource="&rtm;object"/>
 </rdf:Description>

<rdf:Description rdf:about="&owl;versionInfo">
   <rtm:maps-to rdf:resource="&rtm;occurrence"/>
 </rdf:Description>

<rdf:Description rdf:about="&owl;priorVersion">
   <rtm:maps-to rdf:resource="&rtm;association"/>
   <rtm:subject-role rdf:resource="&rtm;subject"/>
   <rtm:object-role rdf:resource="&rtm;object"/>
 </rdf:Description>

 <rdf:Description rdf:about="&owl;backwardCompatibleWith">
   <rtm:maps-to rdf:resource="&rtm;association"/>
   <rtm:subject-role rdf:resource="&rtm;subject"/>
   <rtm:object-role rdf:resource="&rtm;object"/>
 </rdf:Description>
```

```
  <rdf:Description rdf:about="&owl;incompatibleWith">
    <rtm:maps-to rdf:resource="&rtm;association"/>
    <rtm:subject-role rdf:resource="&rtm;subject"/>
    <rtm:object-role rdf:resource="&rtm;object"/>
  </rdf:Description>
</rdf:RDF>
```